JYVÄSKYLÄ STUDIES IN COMPUTING
129

Riikka Ahlgren

Software Patterns, Organizational Learning and Software Process Improvement

JYVÄSKYLÄN YLIOPISTO

JYVÄSKYLÄ STUDIES IN COMPUTING 129

Riikka Ahlgren

Software Patterns, Organizational Learning and Software Process Improvement

Esitetään Jyväskylän yliopiston informaatioteknologian tiedekunnan suostumuksella
julkisesti tarkastettavaksi yliopiston Agora-rakennuksen auditoriossa 2
tammikuun 21. päivänä 2011 kello 12.

Academic dissertation to be publicly discussed, by permission of
the Faculty of Information Technology of the University of Jyväskylä,
in the Agora building, Auditorium 2, on January 21, 2011 at 12 o'clock noon.

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2011

Software Patterns, Organizational Learning and Software Process Improvement

JYVÄSKYLÄ STUDIES IN COMPUTING 129

Riikka Ahlgren

Software Patterns, Organizational Learning and Software Process Improvement

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2011

Editor
Seppo Puuronen
Department of Computer Science and Information Systems, University of Jyväskylä
Pekka Olsbo, Sini Tuikka
Publishing Unit, University Library of Jyväskylä

URN:ISBN:978-951-39-4185-7
ISBN 978-951-39-4185-7 (PDF)

ISBN 978-951-39-4173-4 (nid.)
ISSN 1456-5390

Jyväskylä University Printing House, Jyväskylä 2011

ABSTRACT

Ahlgren, Riikka
Software patterns, organizational learning and software process improvement
Jyväskylä: University of Jyväskylä, 2011, 70 p. (+ included articles)
(Jyväskylä Studies of Computing,
ISSN 1456-5390; 129)
ISBN 978-951-39-4185-7 (PDF), 978-951-39-4173-4 (nid.)
Finnish summary
Diss.

Software process improvement is a necessity in any software organization. This thesis approaches software process improvement from an organizational learning perspective. Process improvements require both learning individuals (i.e. individuals who are learning) and embedding the knowledge into organizational practices and structures. Hence, in this thesis knowledge management and particularly knowledge sharing are investigated in order to find practical means to reach process improvement. Software patterns are analyzed as a concrete tool to support sharing of software design knowledge. This thesis comprises five independent articles. Each article approaches the research topic from different point of view. Consequently, learning and knowledge sharing are examined from individuals', groups' and organizational viewpoints. Multiple case studies were used as a research method and obtained data was both quantitative and qualitative. As a result the research indicates that organizational learning can be supported in software organizations at several organizational levels. Results further indicate that software patterns can be exploited in multiple ways to facilitate knowledge sharing. Patterns can be used as boundary objects to mediate information between individuals and groups. Further, patterns can provide explicit and measurable goals for process improvements in different organizational levels.

Keywords: Software process improvement, organizational learning, knowledge sharing, software patterns

Author's address Riikka Ahlgren
University of Jyväskylä
Dept. of Computer Science And Information Systems
P.O. BOX 35 (Agora)
riikka.ahlgren@gmail.com

Supervisors Professor Samuli Pekkola, Ph.D.
Institute of Business Information Management
Tampere University of Technology

Assistant Professor Eleni Berki, Ph.D
Department of Computer Sciences
University of Tampere

Reviewers Professor Franz Lehner
University of Passau, Germany

Professor Markku Tukiainen, Ph.D
University of Eastern Finland

Opponent Professor Jan Pries-Heje
Roskilde University, Denmark

PREFACE

This thesis summarizes the research that I have conducted in the University of Jyväskylä between 2004 and 2010. The focus of the research has evolved during the research process, starting purely from design patterns and ending up with process improvement and organizational learning.

The research presented in this dissertation begun in 2004 when the Mobile Design Patterns and Architectures project (MODPA) was started at the University of Jyväskylä. The project aimed at improving and rationalizing the development of mobile applications and services with the help of design patterns. In the very beginning of the project it was clear that an organizational view was needed to enable the efficient introduction and adoption of patterns. During the research, knowledge management, learning and knowledge sharing were identified as central research topics. All articles included in this thesis, except the first one, were produced as part of the MODPA –project.

I want to thank all the people in MODPA-project that have given me valuable feedback and food for thought, particularly Jouni Markkula for giving me the opportunity to start the research. Also the MODPA partner companies have been crucial for the research.

I also thank the reviewers Markku Tukiainen and Franz Lehner for the constructive feedback. Thanks also to Pekka Abrahamsson for his commentary on the work. Furthermore I wish to thank Jan Pries-Heje, who volunteered to act as an opponent at the public defense. Special merits go to my supervisor Samuli Pekkola for the can-do attitude, feedback and support that I received. I am also grateful to my second supervisor Eleni Berki for the valuable discussions we had. Both of my supervisors also co-authored some of the included articles. I wish to thank the other co-authors of my papers, Mirja Pulkkinen, Marko Forssell and Jari Penttilä. Thanks also to the rest of ITRI-personnel for the refreshing discussions during coffee breaks.

The research reported in this dissertation has been financially supported by the National Technology Agency of Finland (TEKES), the MODPA partner companies Nokia, Yomi Software, SESCA Technologies and Tieturi and by the Graduate School in Computing and Mathematical Sciences (COMAS).

I am grateful for the Department of Computer Science and Information Systems for providing me the facilities for finalizing the dissertation. I also thank the publishers of the original papers for giving me the permission to reprint the articles as part of my thesis. I wish also to thank my colleagues at Visma for making me laugh. And thanks to all the other wonderful people who have supported me during these years.

Jyväskylä 7.12.2010
Riikka Ahlgren

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

In order to stay competitive in their business, software organizations constantly search and adopt faster and cheaper ways to work. Process improvements in daily work are crucial to gain savings (Rico, 2004; Zahran, 1998; van Solingen, 2009), which gives an initial motivation for this research. Motivation for this study is visible also in statistics. In 2009 only 32% of IT projects were delivered on time, on budget and with required features (Standish Group, 2009). These results indicate that improvements in software projects and processes are definitely needed.

Turning improvement ideas into actual changes in development process does not happen automatically but requires communication, coordination and knowledge management (Alavi and Leidner, 2001; Niazi, Wilson & Zowghi, 2006; Dybå, 2005; Heikkilä, 2009). This applies particularly in knowledge intensive software industry, where employee turnover can have severe consequences (Seleim, Ashour & Bontis, 2007). Consequences of losing key personnel can be decreased with business processes that preserve valuable knowledge in the company, even if personnel changes. However, reality in software organizations does not always support these processes: project schedule pressure can limit the time for learning (Nan & Harter, 2009), structures for knowledge sharing often are inadequate (Mathiassen & Pedersen, 2005) and individuals do not participate in the change (Abrahamsson, Salo, Ronkainen & Warsta, 2002; Nasir, Ahmad & Hassan,2008). This motivates a closer study of the organizational practices and tools that can facilitate learning and knowledge sharing at individual, team or organizational levels.

Generally, business processes guide organizational tasks and activities in a common direction. They collect multiple practices into coordinated chains of actions (Wenger, 2003). In software organization, *software development process* describes the sequence of steps required to develop or maintain software (Humphrey, 1995). A software development process is generally composed of specification, implementation, validation and evolution phases (Sommerville, 2001). Actual tasks and relations of the phases vary according to specific process models. Commonly used models are, for example, the waterfall and spiral

models (Boehm, 1988; Pressman, 2000). In recent years particularly agile process models have gained increasing interest both in research and in the software industry (e.g. Abrahamsson et al., 2002; Suscheck & Ford, 2008; Wirfs-Brock, 2009).

Regardless of the process model, continuous process improvement is a necessity in the swiftly evolving software business (Niazi, 2006; Oktaba & Piattini, 2008; Trienekens, Kusters, van Genuchten & Aerts, 2008). *Software processes improvement* (SPI) is a constant effort that aims at producing quality products in an efficient manner by utilizing the best practices of the field (Humphrey, Over, Konrad & Peterson, 2007). The reasoning of software process improvement is based on the assumed relationship between software process maturity and product quality (Humphrey, 1989; Schönström, 2005). In a mature software process, the involved people, various development methods and technologies are combined into an effective and efficient collection of practices (Humphrey, Kitson & Kasse, 1989).

In business terms, a mature process and a consequent high product quality is expected to lead to a higher return on investment (ROI) (Krasner, 2001; van Solingen, 2004). Furthermore, fewer errors in products, i.e. a higher product quality, should lead to a positive impact on customer satisfaction (Zahran, 1998). In addition, mature processes enable removing some of the rework and facilitate the introduction of new technologies (Harter, Krishnan & Slaughter, 2000; Zahran, 1998). Thus, mature processes are expected to shorten the product's time to market. Several research results support these expectations (Diaz & Sligo, 1997; Haley, 1996; Harter et al., 2000; Hollenbach, Young, Pflugrad & Smith, 1997; Niazi, Wilson & Zowghi, 2005).

However, there are also evident problems with SPI initiatives. These often relate to SPI costs (Leung, 1999; van Solingen, 2004). Other critical barriers in SPI are time pressure, inexperienced staff, poor planning, lack of SPI awareness and lack of support and resources (Brietzke & Rabelo, 2006; Nasir et al., 2008; Niazi, 2009; Staples, Niazi, Jeffery, Abrahams, Byatt & Murphy, 2007). In addition to SPI related problems, failures related to learning are also well known in the software business. Learning obstacles can include the developers' lack of business understanding, inadequate encouragement for learning, unsuitable organizational design and overly high expectations (Lyytinen & Robey, 1999; Mathiassen & Pedersen, 2005). To overcome these barriers, systematic learning is required from all people involved (Ravichandran & Rai, 2003).

In the past, learning has been studied in many contexts. Previous research has focused for example on individual learning, experiential learning (e.g. Kolb, 1984; Turner, Keegan & Crawford, 2000) and situational learning (Collin, 2005; Kim, 1993). Since the focus of this research lies in organizations and their practices, here we concentrate on organizational learning and its implications in software development. *Organizational learning* (OL) is an umbrella concept that covers knowledge adoption, knowledge sharing and creation of new knowledge (Argote, 1999; Huber, 1991). It has been defined as "the process of change in individual and shared thought and action, which is affected by and embedded in the institutions of the organization" (Vera & Crossan, 2003). However,

organizational learning does not equate with the cumulative learning of an organization's members (Huysman, 2000), although individual learning is a prerequisite for organizational learning (Kim, 1993; Nonaka, 1994). In individual learning, the individuals not only need to grasp the contents (know-how), but also to understand and apply the reasoning (know-why) (Kim, 1993).

Many other disciplines are also related to organizational learning (Easterby-Smith, 1997). To distinguish between these related concepts, organizational learning can be considered as focusing on learning processes (Huber, 1991). Moreover, organizational learning views the organization as a complex and living system (Robey, Wishart & Rodriquez-Diaz, 1995) and links both knowledge and action into same processes (Crossan, Lane & White, 1999). *Learning organization*, in turn, forms an ideal entity that has the capability to adopt those learning processes in practice (Easterby-Smith & Lyles, 2005). *Organizational knowledge* describes the form and nature of the knowledge that is possessed by the organizations (Easterby-Smith, Crossan &Nicolini, 2000).

A crucial part of organizational learning is the collection of past information and knowledge, which is exploited for future actions (Walsh & Ungson, 1991). This collection, which is interpreted for example in organizational processes and structures, forms *organizational memory* (OM) (Perez Lopez, Montes Peon & Vasquez Ordas, 2005; Walsh & Ungson, 1991). To emphasize the dynamic nature of organizational memory, here we adopt a definition according to which organizational memory is a process consisting of four sub-processes. The processes of knowledge acquisition, retention, maintenance and retrieval are on constant interaction with the organization's structures, practices and tools (Stein, 1995). To collect and utilise the knowledge that is stored with the organizational memory requires *knowledge management* (KM).

Knowledge management relies on the differentiation between *data*, *information*, and *knowledge*. They can be distinguished by the base of the knowledge (Bhatt, 2001). *Data* is a set of discrete, objective facts about events, often described in an organizational context as structured records of transactions (Davenport & Prusak, 1998). *Information* is often characterised as a "message", usually in the form of document or another visible or audible format (Alavi & Leidner, 2001; Davenport & Prusak, 1998). *Knowledge* then derives from information. Information is descriptive by nature, while knowledge is associative (Kock, 1999; Mathiassen & Pedersen, 2005).

In the past, research of knowledge management has been versatile and even controversial (see e.g. Jennex & Olfman, 2004; Prusak, 2001). Many scholarships exist; some approach KM from technology-related perspectives (e.g. Kankanhalli, Tan & Wei, 2005; Lindvall, Rus & Sinha, 2003) while others are interested in organizational aspects (Davenport & Prusak, 1998; Easterby-Smith & Lyles, 2005; Kakabadse, Kakabadse & Kouzmin, 2003; Nonaka & Takeuchi, 1995). From an organizational perspective, KM is defined as "a method that simplifies the process of sharing, distributing, creating, capturing and understanding of a company's knowledge" (Davenport & Prusak, 1998).

Regarding the organizational aspects of knowledge management, three approaches are commonly adopted. Firstly, there is the knowledge capturing approach that focuses on the distribution and efficient use of personalized knowledge (Hansen, Nohria & Tierney, 1999). Secondly, there is the community-based approach that regards knowledge as socially constructed. This emphasises interaction and dialogue as a means to share knowledge, and highlights the importance of social ties and trust as success factors (Mathiassen & Pourkomeylian, 2003). A third, the cognitive approach, concentrates on the codification of experiences, storing resulting artefacts and their efficient reuse (Mathiassen & Pourkomeylian, 2003; Ravichandran & Rai, 2003). This thesis adopts the first approach and thus focuses on efficient distribution and sharing of codified knowledge. In particular, patterns are studied as a format of codified knowledge.

A central process in knowledge management is *knowledge sharing*. It includes both giving and receiving information in a certain context (Sharrat & Usoro, 2003). Sharing can be done on an ad-hoc basis, such as during coffee breaks, but it is more efficient when organized (Lindvall et al., 2003). Boland and Tenkasi (1995) use term "perspective making" to describe knowledge sharing and particularly how knowledge is developed and strengthened in a community. Respectively, the term "perspective taking" is then used to explain the communication required to acknowledge other communities' knowledge (Boland & Tenkasi, 1995). Davenport and Prusak (1998) in turn use the term "knowledge transfer" to describe any kind of communication, whether it is managed or not (Davenport & Prusak, 1998). This thesis emphasizes communications in knowledge sharing and hence uses the definition by Davenport and Prusak (1998). To better depict the two sides of the activity, i.e. giving and receiving, we use the term knowledge sharing.

In software organizations the central knowledge to have and to exploit is business knowledge, technical skills and design knowledge (Ketola, 2002; Terveen, Selfridge & Long, 1993; see also Curtis, Krasner & Iscoe, 1988). In this thesis *design knowledge* is knowledge that combines business visions and technical solutions into a purposeful entity, adding value to a single solution by giving it a context and purpose. The major characteristic of this kind of intellectual capital is that it can only "run" as fast as the people who carry it and hence, organizations become vulnerable unless this knowledge is not efficiently managed. Failure to manage design knowledge can result in low-quality designs, late and costly error detection, late deliveries and personnel frustration (Terveen et al., 1993). Therefore, capturing and adopting design knowledge is essential for any software organization. In the software development community, *software patterns* have been adopted as a format for capturing the design knowledge.

The concept of patterns was originated in architecture (Alexander, Ishikawa, Silverstein, Jakobson, Fiksdahl-King & Angel, 1977) and was brought to software development when the book "Design Patterns" was published in 1995 (Gamma, Helm, Johnson & Vlissides, 1995). A classical definition for a pattern is

"a three-part rule, which expresses the relation between a certain context, problem and solution" (Alexander, 1979). Hence, software patterns are pieces of software design that have worked well in the past, that have been documented in a certain manner and that can be used in similar situations in the future. They exist in a particular format that enables exact communication of the subject (May & Taylor, 2003).

This thesis studies patterns from the perspectives of organizational learning and process improvement. A basic precondition for process improvement is that people are able to understand why and how they do the things that they do during the development, so that the actions can be intentionally repeated when new problems emerge (Dybå, 2001; Slaughter & Kirsch, 2006). If an activity is performed without understanding, such as by slavishly following guidelines, improvements can be hard to identify. In consequence, a tentative research question for this thesis is composed as

"How can organizational learning be supported in software organization?"

This question is approached from both individual and organizational perspectives. First, software developers' typical working activities are characterized. Based on those the thesis then suggests improvements for organizational practices. As a main contribution, an analysis of organizational learning in software development is produced. This analysis is needed in order to actually improve the software development process. Further, to embed the improved processes into daily development work, the understanding of good and efficient development practices must be shared in software organization. Sharing of software design knowledge highlights the need of knowledge management and hence, raises interest in software patterns. Consequently, the second tentative research question for this thesis is

"*Can software patterns serve as a tool for knowledge sharing in software development?"*

To find the answer to this second question, the characteristics and use of software patterns are examined. Case studies are conducted to reveal the concurrent and possible use of software patterns as a tool for communication, learning and knowledge sharing. The main contribution to the second research question is an analysis of software patterns as a tool for communication, learning and knowledge sharing.

Taking together the concepts presented above it is arguable that SPI, organizational learning and knowledge management are tightly connected. Furthermore, in order to succeed, SPI requires investments in organizational learning. Organizations must be able to capture, share and adopt the knowledge of their employees. Organizational memory is required to enable the organization to learn. In software organization, this means that design knowledge must be preserved in such place and in such format that it can be found, understood and

used in daily development work. In turn, the building and use of organizational memory is dependent on knowledge management and particularly on knowledge sharing. Here, software developers are in key positions. Knowledge sharing requires organizational support in order to realize within organizations. This, in turn, is a task for company management. This chain of concepts and their assumed dependencies forms the skeleton for this thesis, illustrated in figure 1.

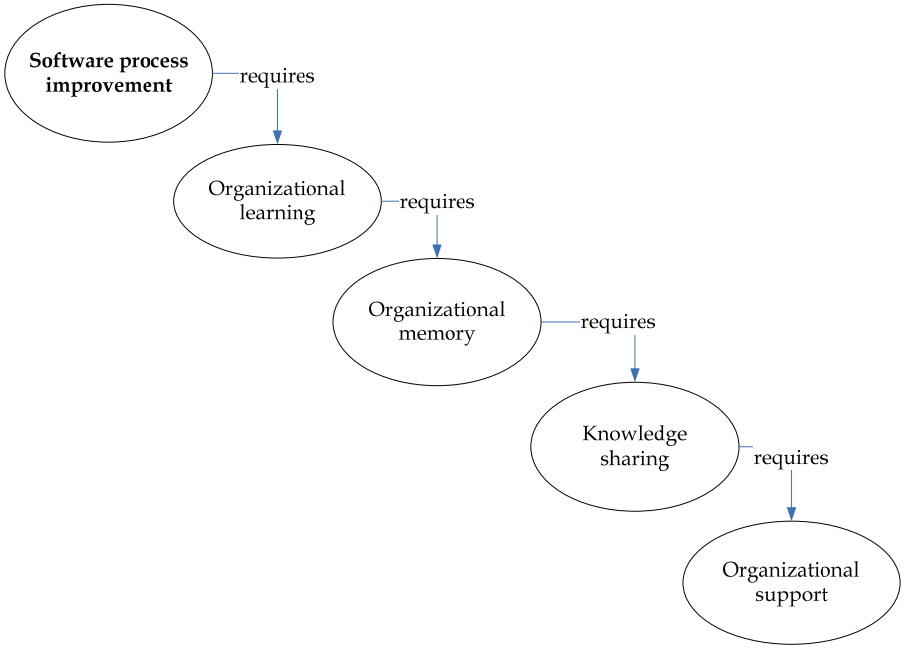

FIGURE 1 Construction of the thesis

Figure 1 will be analyzed and validated in the following chapters. The validation is based on literature analyzing characteristics and major mechanisms of both SPI and organizational learning. Furthermore, examples are given of contents of organizational memory and the sharing of that knowledge. This will then lead to software patterns, which are examined with case studies as a particular tool for knowledge sharing.

2 THEORETICAL BACKGROUND

The following sub-sections present the theoretical background and relevant literature for this research. Aims, processes and models for software process improvement are first outlined. Next, main characteristics and components of organizational learning in software development are presented. There follows an analysis of knowledge management and knowledge sharing in particular, in connection to SPI. Finally, software patterns and their use as a communication tool are investigated.

2.1 Software process improvement as a learning process

No organization stays competitive by endlessly repeating the same tasks and activities in the same way (Niazi, 2009; van Solingen, 2009). In order for software organizations to be able to deliver larger and more complex software systems, their business processes need to be refined and continuously improved (Niazi, 2009; Trienekens et al., 2008). The aim of process improvements is to make the processes more efficient and, eventually, to raise the product quality (Zahran, 1998). Since process improvement requires time and money, it is focused on the most central business processes. In software organization, the main target of the improvement initiatives is hence software development processes.

The three key elements in efficient process improvement are planning, implementation and communication (Humphrey, 1989). For efficient implementation, resourcing and correct roles are crucial (Johansen & Pries-Heje, 2007). There must be enough time for the SPI team to do their work and they need to have the skills and authority to perform their jobs. Aligning improvement goals with business goals is also central (Dybå, 2005; Conradi & Fuggetta, 2002). This ensures that SPI is economically profitable for the company. In turn, communication is needed to inform about the improvements, targets and practices, to involve the people in the actions and to gain feedback on the process. Mentor-

ing and training are particularly important communication practices (Rainer & Hall, 2002). These enable close interaction between people, which further facilitates knowledge adoption and understanding.

Efficient and useful processes require shared visions and practices. Hence, communication of the concrete, shared vision is central in SPI (O'Hara, 2000). The involved people in the organization must identify and acknowledge a common conception about their working environment and the work processes. In other words, the main idea of an SPI initiative is to create and share knowledge between individuals, teams and departments, i.e. across organizational boundaries (Mathiassen & Pourkomeylian, 2003; Wenger, 2003). These organizational boundaries are formed by groups of individuals, teams or departments who create a community by engaging in a common process, by having a common knowledge base or other shared interest (Wenger, 2003). It is important to note that the communities do not necessarily follow the official organizational groups, but are merely defined by the shared knowledge (Star & Ruhleder, 1996; Wenger, 2003). Specific artefacts, i.e. boundary objects, can then be used to facilitate the boundary crossing (Wenger, 2003). The boundary objects are artefacts that reach several communities and satisfy their information requirements (Bowker & Star, 1999; Star & Griesemer, 1989).

An efficient process requires that the people have a mutual understanding and agreement on the process targets, the required actions and the way the actions should be performed (Akgün, Lynn & Reilly, 2002). To create this shared understanding, process descriptions and routines need to be explicitly described (Akgün et al., 2002; Senge, 1990). The explicit procedures can function as boundary objects and enable people to communicate and coordinate their actions across processes (Wenger, 2003). Crossing processes and other organizational boundaries requires knowledge sharing; hence, boundaries create opportunities for learning.

Opportunities for learning are essential when developing complicated software systems, where many different expertises are needed. In software development, the need for knowledge and need for communication are constant, whether it is for complexity handling, problem solving or design decisions. Thus, software development is a highly knowledge-intensive activity. Therefore, approaches to knowledge management are essential in SPI initiatives (see e.g. Mathiassen & Pourkomeylian, 2003; Ravichandran & Rai, 2003). In fact, SPI can be seen as a special form of knowledge creation, sharing and management (Mathiassen & Pourkomeylian, 2003). Further, knowledge management provides a systematic approach to various collaboration, communication and coordination needs that are present or emergent in software development (Mathiassen & Pourkomeylian, 2003). Indeed, knowledge management has gained a vast research interest as an approach to SPI. The KM perspective has covered, for example, the reuse of components (Kucza, Nättinen & Parviainen, 2001), knowledge processes (Schönström, 2005) and knowledge transfer portfolios (Slaughter & Kirsch, 2006). This thesis aims to build on the knowledge management perspective on SPI, focusing particularly on organizational learning

during the software development. Software patterns provide one way to start the SPI efforts "bottom-up", for they can be used as boundary objects to mediate development knowledge. Further, they can provide a concrete initiative that is claimed for successful SPI, since they provide support for expertise reuse, and hence can facilitate organizational learning (Conradi & Fuggetta, 2002).

2.1.1 Models for process improvement

Successful SPI initiatives need goals. These goals need to be aligned with the particular organization and its business strategy (Dybå, 2005). Apart from the specific SPI goals, efficient software processes have other common features. These include

1. controlling development costs and schedule predictability,
2. responding to changing needs,
3. minimizing development costs and schedules,
4. scaling from small to very large systems and
5. predictable production of quality products (Humphrey et al., 2007).

To gain these goals, a number of process models have been presented. Process models provide frameworks and plans for producing software, while at the same time improving developers' capabilities to produce better products (Humphrey et al., 1989; Lindvall & Rus, 2000). Some models stress the improvement cycle while some are merely collections of the best practices in software development. A well-known model for continuous process improvement is the Deming cycle illustrated in figure 2. This plan-do-check-act (PDCA) cycle consists of four activities that are repeated to achieve ever-higher levels of quality (Deming, 1994). In the PDCA cycle, a change is first planned and then implemented. Afterwards the results are checked and lessons learned are analyzed. Finally, the change is applied or abandoned, depending on the results and their analysis.

In general, process models and frameworks can be used to assess an organization's maturity, and to identify and prioritize the areas for improvement (Saiedian & Chennupati, 1999). Models and frameworks rely on process structures; by breaking down processes into sub-processes and separating practices, they make processes more manageable and measurable. Commonly known frameworks for SPI are for example IDEAL^{SM}, SPICE -standard (Software Process Improvement and Capability Determination, also known as ISO/IEC 15504) and CMMI (Capability Maturity Model Integrated) (Dorling,1993; McFeeley, 1996; Zahran, 1998). Both SPICE and CMMI are used in many types of software organizations (Zahran, 1998), though CMMI has been more popular in America while SPICE has been favoured in Europe (Stelzer & Mellis, 1998).

IDEAL^{SM} is based on CMM. It describes a complete path for software process improvement, comprising steps for planning, conducting and managing SPI (McFeeley, 1996). Intention of the IDEAL model is to give guidance on how to execute an improvement program and how to turn the assessment re-

sults into actions. The model consists of initiating, diagnosing, establishing, acting and learning phases, all of which include different activities for guiding the improvements.

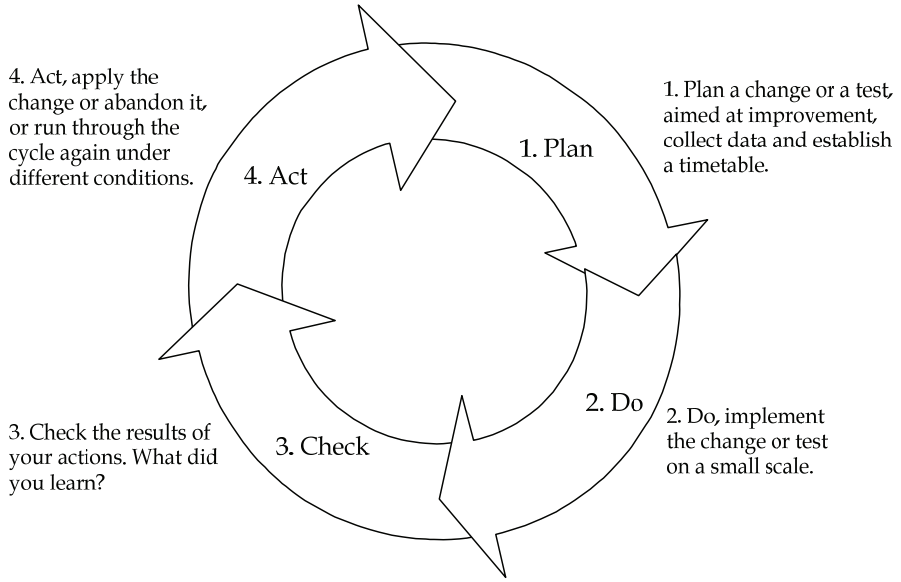

FIGURE 2 Deming- cycle for process improvements (Deming, 1994)

Another widely used SPI framework is SPICE (Software Process Improvement and Capability dEtermination), officially ISO15504 (El Emam, Drouin, & Melo, 1998). It was developed in mid 1990s with the purpose to collect the best features of existing software assessment methods (Dorling, 1993). SPICE is composed of two dimensions: process dimension and capability dimension. The process dimension divides the processes into five categories: customer-supplier, engineering, project, support and organization. Capability dimension then in turn comprises of six capability levels ranging from zero to five, which describe the process overall potential. The levels are described as ad hoc (level 0), performed, repeatable, defined, managed and optimized (level 5). (Dorling, 1993; El Emam et al., 1998)

A third commonly-used approach for SPI is CMMI (Capability Maturity Model Integrated). CMMI is a collection of best practices and objectives that are used for assessing the maturity of software processes in a company. In CMMI, all development related processes are divided into 22 process areas. A process

area is a cluster of practices, which are fundamental for making improvement in that specific area. The process areas are, for example, project planning, requirements management and configuration management. Process areas are then further grouped into four categories: process management, project management, engineering and support. To assess the software organization's maturity according to the CMMI, all the organizational software processes are ranked according to five maturity levels. These maturity levels are listed in the following table 1. With CMMI, software organizations can progressively improve their processes, first by achieving control at the project level and then continuing to the organization-wide process improvement (CMMI for Development, 2006; Zahran, 1998).

TABLE 1 Maturity levels in CMMI –process model (Kulpa & Johnson, 2008).

Maturity level	Description
1 Initial	Ad-hoc development, no defined process structures.
2 Managed	Basic project management in place and followed.
3 Defined	Organizational way of doing business. A set of standardized, tailored processes is in use.
4 Quantitatively managed	Processes are controlled by statistical and other quantitative techniques.
5 Optimizing	Processes are continuously improved based on common causes of variation.

In this thesis, CMMI is adopted as a reference model in selected cases because the model is commonly adopted in software organizations (e.g. Carnegie Mellon University, 2009; Ngwenyama & Nielsen, 2003; Saiedian & Chennupati, 1999), it has a clear structure and wording and thus its goals are easy to understand and adopt, and because it is easily accessible in the internet[1].

2.1.2 Models summarized

Regardless of SPI model or framework, all have a common concern for software quality and process management. A great advantage of the models is that they define the targets and practices that need to be accomplished in order to reach certain improvements. Thus, a model provides a shared vision for the process improvement. They do not, however, say exactly how the practices are to be implemented, on what level they should be implemented to add value to the company, or how the required changes are introduced into the organization. These are often the particular topics where software managers need guidance (Herbsleb, Zubrow, Goldenson, Hayes & Paulk, 1997; Wilkie, McFall & McCaffery, 2005). Furthermore, a major drawback of the models is a lack of consideration of the impact of various contextual factors, e.g. reward systems and organizational commitment (Abrahamsson et al. 2002; Ravichandran & Rai, 2003).

[1] www.sei.org

From the business success perspective it is important to find a balance between the discipline represented by the process models and product innovation rising from the development team's expertise (Conradi & Fuggetta, 2002). Since the essence of process development is to identify both the process steps where rigor is needed and the places where creativity can flourish (Conradi & Fuggetta, 2002), it is important for an organization to first identify and define its business needs and goals, and only then select the SPI model that best fits the situation (Pries-Heje & Johansen, 2010; Saiedian & Chennupati, 1999). Furthermore, since individuals and teams are the ones who make the SPI happen, merely breaking down the processes into a process model will not immediately bring the business benefits of SPI (Humphrey, 1989; Niazi, 2009; Trienekens et al., 2008). Therefore, whatever the chosen model, developers, testers, project managers and, eventually, the software organization must start doing things differently.

2.1.3 Organizational needs in successful SPI

The role and importance of organizational issues in SPI has been studied by many researchers within different research settings (e.g. Abrahamsson et al., 2002; Gasston & Halloran, 1999; Johansen & Pries-Heje, 2007). This has further raised the overall interest and importance of the non-technical factors in SPI (Dybå, 2001). It has been reported that SPI initiatives can be supported, particularly with employee participation at the SPI-actions, an organization's overall concern of SPI measurements and with the ability to implement the measurements (Dybå, 2001). Thus, these and also other research results (Kock, 1999; Pries-Heje & Johansen, 2010; Robey et al., 1995; Segal, 2001) heavily emphasize organizational factors in planning and implementing SPI-initiatives.

To enable the continuous software process improvement, systematic learning in the organization is a necessity (Mathiassen, Nielsen & Pries-Heje, 2002; Ravichandran & Rai, 2003; Seigerroth & Lind, 2006). Research results indicate that SPI can benefit from the organizational learning paradigm. Essential factors in successful SPI are, for example, providing a context for learning through procedure implementation and management initiatives, having a method to examine learning capabilities and establishment of norms, and having a culture that encourages learning (Gasston & Halloran, 1999). Taken together, organizational learning is needed in order to gather, package and share the lessons among employees. Furthermore, organizational learning is required to embed the improved working practices and high-quality software solutions as organizational practices. Hence, by making knowledge available for the organization, the learning processes also allow the software processes to improve.

2.2 Organizational learning as a collective process

As stated above, learning processes and software processes improvements are necessities in order to increase software product quality (Niazi, 2009; Perez Lopez et al., 2005). Nevertheless, in software development, quality is heavily dependent on developers' skills (Cockburn, 2001; Humphrey, 1989; Koc, 2007). In addition, the best people are always a scarce resource, and not even the best people can manage everything on their own. Furthermore, even the best people can leave and take their talents with them. Thus, learning from each other and learning from one's own experiences are crucial in producing high quality software (Collin, 2005; Holden, Smith & Devins, 2003; Lyytinen & Robey, 1999). However, this requires that individuals become aware of efficient practices and know-how, and that they consequently apply these to their daily working practices. The adopted knowledge then constructs mental models, i.e. assumptions, habits, attitudes and visions of how things are (Senge, 1990). These mental models are combinations of explicit knowledge and implicit skills, i.e. tools that allow the individuals to perform the necessary actions (Senge, 1990).

Sharing of individually constructed mental models and visions is essential when transforming individual learning into organizational learning (Kim, 1993; Senge, 1990; Huysman, 2000; Bennet & Bennet, 2008). Further, as noted earlier, organizational learning requires more than a group of learning individuals. In order to create organizational knowledge and to enable organizational learning, the individual learning results need to be shared and adopted into organizational values, processes and practices. This is referred to as organizational double-loop learning (Argyris and Schön, 1978; Kim, 1993). In summary, since knowledge sharing depends on both organizational and individual factors, the connection between organizational learning and actual changes in working processes, i.e. SPI, is not evident (see. e.g. Alavi & Leidner, 2001; Huysman, 2000; Irani, Sharif & Love, 2009), but deserves closer attention.

2.2.1 Organizational memory and types of knowledge

For the organization to be able to exploit the knowledge they have, the knowledge must be stored and organized in an appropriate manner. To describe the different characteristics and sources of knowledge in organizational communications, it is often divided into tacit and explicit knowledge (e.g. DeSouza, 2003; Mathiassen & Pedersen, 2005; Nonaka & Takeuchi, 1995). Knowledge can be held by individuals or it can be embedded in the organizational structures (see eg. Mathiassen & Pedersen, 2005). To further clarify this versatility of knowledge, figure 3 gives examples of knowledge types in organizational context.

In the upper left corner of the figure there is explicit knowledge held by individuals, i.e. proprietary knowledge or codified experience (Dybå, 2001; Mathiassen & Pedersen, 2005). This "embrained" knowledge could be for example software patents and Post-it notes of application details. The upper right corner represents explicit knowledge held collectively by the organization, and

includes models, prototypes, process descriptions, or other formal routines and shared pieces of encoded knowledge (Dybå, 2001).

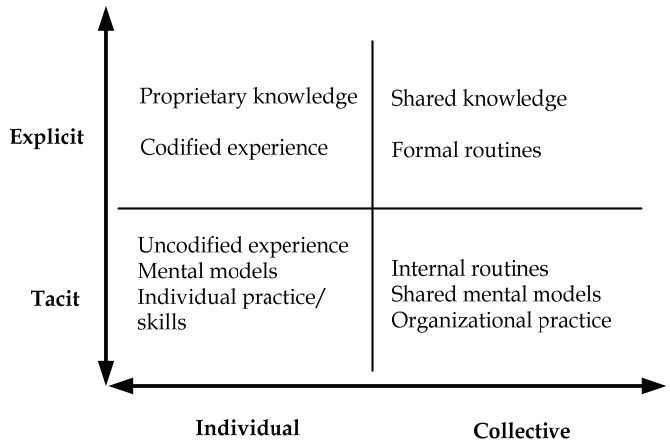

FIGURE 3 Knowledge in organizations (Dybå, 2001)

In software organizations, for example, this type of knowledge includes software mock-ups or design diagrams. Tacit knowledge possessed by individuals (i.e. embodied knowledge) is situated in the lower part of the figure. That includes mental models and attitudes and various individual skills. Moving to the right, internal routines and organizational practices represent collective but tacit knowledge, i.e. embedded knowledge. For example, unspoken rules about development practices belong to this category. (Dybå, 2001; Mathiassen & Pedersen, 2005)

The storing, organizing and retrieving of organizational knowledge is often called organizational memory (Stein, 1995; Walsh & Ungson, 1991). Organizational memory can be seen from a functional process perspective (Alavi & Leidner, 1999) or from technical perspective, where IT systems are central (Jennex & Olfman, 2002). Regardless of the perspective, the core issue is that the knowledge can exist in various formats – e.g. databases and e-mails – and it can be explicit or tacit. Consequently, for knowledge-intensive work like software development, a well managed organizational memory is a necessity. The organizational activities that are used to manage organizational memory can include decision-making, problem solving, coordinating, controlling and planning (Alavi & Leidner, 1999). In software development, lack of explicit knowledge can slow down new employees' learning (Akgün, Lynn & Reilly, 2002). For example, if a release procedure or instructions for a packaging tool are only in tacit format, the old developers know by heart what has to be done in order to release a product, but a new developer has nothing with which to start. Supporting material, e.g. process description or a user manual for the tool, can significantly facilitate learning (Zhong & Majchrzak, 2004). Hence, the need for a different kind of knowledge is also evident in a software development context.

2.2.2 Knowledge sharing

Knowledge acquisition, knowledge sharing and knowledge integration are significant and time-consuming activities in software development (Korkala & Abrahamsson, 2007; Walz, Elam & Curtis, 1993). The issues to be handled are complex, and activities for knowledge management are needed on a daily basis (Korpela, Mursu & Soriyan, 2002). Therefore, specific organizational structures, or "experience factories", have been suggested to support knowledge reuse (Basili & Caldiera, 1995). They were to facilitate collective learning by analyzing and packing the lessons learned for the rest of the organization. However, not even experience factories can work without the individuals who process the relevant knowledge.

Sharing knowledge creates new knowledge. This, in turn, is a prerequisite for innovation and learning (Alavi & Leidner, 2001; Calantone, Cavusgil & Zhao, 2002; Nonaka & Takeuchi, 1995). Nonaka & Takeuchi introduced a theory of organizational knowledge creation to describe the processes that are central in knowledge sharing (Nonaka, 1994; Nonaka & Takeuchi, 1995). It is widely used in information systems science (e.g. DeSouza, 2003; Komi-Sirviö, Mäntyniemi & Seppänen, 2002; Ravichandran & Rai, 2003; Schönström, 2005) and hence it is also adopted in this thesis. The continuous and dynamic interaction between tacit and explicit knowledge, where knowledge is created and shared, is called SECI-process (illustrated in figure 4). It consists of Socialisation, Externalization, Combination and Internalization (Nonaka & Takeuchi, 1995).

An individual's tacit knowledge is the basis of organizational knowledge creation. Tacit knowledge is hard to communicate because of its personal, experiential nature. During socialization, these experiences and mental models are shared in discussions and dialogue (Nonaka & Takeuchi, 1995; Senge, 1990).

Socialization is often enabled by routines and shared experiences (Nonaka & Toyama, 2003). In the externalization process, tacit knowledge is made explicit. In particular, cooperative dialogue that includes both dialogue and discussion is crucial, since it fosters conceptualization of the knowledge (Nonaka, 1994; Nonaka & Takeuchi, 1995; Senge, 1990). In the combination process in turn, explicit knowledge is gathered, edited, integrated and diffused in the organization. This process is central in order to make the knowledge understandable, structured and conceptualized (Nonaka & Takeuchi, 1995). In the internalization process, the explicit knowledge, which is shared throughout the organization, is then converted back into tacit knowledge. This is needed for the individuals to get practical use of the knowledge. In summary, in the process of knowledge sharing the organizational knowledge is shared between people as tacit knowledge, made explicit in dialogue, shared in an explicit format by conceptualizing it and then transformed again into tacit knowledge when adopted in individuals' work tasks. Hence, these four processes are interdependent and intertwined to a large extent. This means that the processes rely on, contribute to, and benefit from other processes (Alavi & Leidner, 2001).

Therefore, sharing of both tacit and explicit knowledge is critical in organizational knowledge creation and further to enable organizational learning (Nonaka, 1994; Nonaka & Takeuchi, 1995; Nonaka & Toyama, 2003; Sharratt & Usoro, 2003).

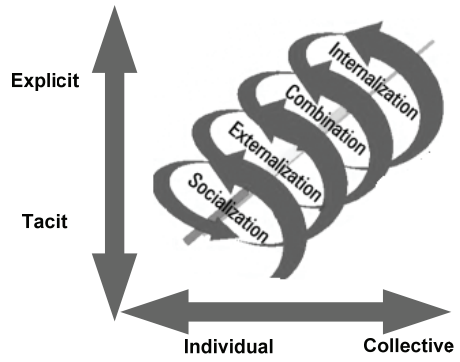

FIGURE 4 SECI-process (Nonaka & Takeuchi, 1995)

2.3 Knowledge management in SPI

Software organizations generally possess different types of knowledge that are represented in various ways. This diversity of knowledge makes sharing it a challenging task. However, as it has been described above, knowledge and knowledge sharing is crucial in software development (Zdrahal, Mulholland, Domingue & Hatala, 2000; Walz et al., 1993). Typically, building large programs involves handling and coordinating multiple sources of knowledge (Kettunen, 2003). The knowledge may relate for example to the intended application domain of the program, or to technical issues, such as the use of software engineering tools or techniques (Rus & Lindvall, 2002; Waterson, Clegg, Axtell, 1997). Furthermore, the size of many large applications and distributed development projects contributes to the fragmentation of related knowledge (Kotlarsky & Oshri, 2005; Waterson et al., 1997). In addition, due to the vast size of the software, often there are not many people (if any) who can master the whole system. Hence, knowledge management and particularly knowledge sharing, learning and crossing knowledge boundaries become essential for successful software development.

Past research presents success factors for knowledge management and knowledge management systems (e.g. Alavi & Leidner, 2001; Jennex & Olfman, 2004; Wong, 2005). Among other factors, the prevailing organizational culture has an impact on knowledge management and particularly on knowledge

sharing (Iivari & Huysman, 2007; Jennex & Olfman, 2004). Collaborative organizational culture and knowledge socialization, externalization and internalization are positively related (Lee & Choi, 2003). Hence, explicit and shared organizational attitudes and efficient practices are fundamental in order to avoid the various knowledge sharing problems (Aurum, Daneshgar & Ward 2008; DeSouza, 2003; Calantone et al., 2002). A typical knowledge management problem in software development relates to expertise and its consequences. Being acknowledged as an expert of a particular tool, technology or of a certain part of the software is not always a positive thing for a software engineer, as it can be seen limiting the developers' future tasks and career instead of advancing them (DeSouza, 2003). Therefore, this forms a major barrier for effective knowledge sharing. Another challenge is the perceived difficulty in categorizing and explaining tacit development knowledge (Alavi & Leidner, 2001; DeSouza, 2003), as developers often find it troublesome to explain what they do and how they solve particular problems in their work (Aurum et al., 2008). Also, people possessing other roles in the development team, e.g. quality assurers, can have great difficulties in communicating specific task-related knowledge that is not explicitly described, but which, for example, has been learned by experience (Aurum et al., 2008).

In addition to organizational culture, the available tools and techniques can be inadequate in addressing effective knowledge management (Aurum et. al, 2008; Jennex & Olfman, 2004). For example, many alternative communication channels (e.g. knowledge system, e-mail, peer-to-peer discussions. etc.) can complicate knowledge sharing instead of simplifying it (Alavi & Leidner, 2001; DeSouza, 2003). Further, when standard development processes leave only little room for experimentation, tailoring the processes to the current limitations and realities is challenged (Kulpa & Johnson, 2008). Acknowledging the barriers of knowledge sharing, it is not surprising that a recent study indicates that knowledge management in software organizations is quite immature and that often the preferred development approach is simply "learning by doing" (Aurum et al., 2008). However, since software development is extremely knowledge dependent, efficient knowledge management can bring great benefits not only to individuals, but also development teams and the whole software organization.

The benefits of knowledge management which profit the entire software organization are, first of all, commercial. These include, for example, savings in development costs and shortened development time (Rus & Lindvall, 2002). The development team in turn, can avoid repeating their previous mistakes, and even mistakes made by other teams and individuals. Additionally, efficient knowledge management enables the development teams to better exploit past successes. Creating process models in turn facilitates mutual communication within and between teams (Alavi & Leidner, 1999; Rus & Lindvall, 2002). A well-organized knowledge management can also benefit software development processes, thus increasing product quality (e.g. Jennex & Olfman, 2002; Montoni, Zanetti & da Rocha, 2008; Rus & Lindvall 2002). Staff participation and

communication are increased, which can further reduce problem-solving time (Alavi & Leidner, 1999). Furthermore, knowledge management supports the creation and utilization of product and project memory, for example with version control or with explicit change management practices (Rus & Lindvall, 2002). This in turn can further increase productivity (Jennex & Olfman, 2002). Process improvement and individuals' learning in turn can be supported by capturing and maintaining knowledge. Central activities can include collecting project data or lessons learned from previous projects, enabling them to be used for future predictions. (Rus & Lindvall, 2002)

Taken together, an efficient knowledge management provides a vast selection of improvement targets in software development. But to gain the benefits, the knowledge gathered needs to be processed in a relevant context. Processing the knowledge is about transforming the knowledge format and type in order to assimilate, adopt, share and finally embed it into the organization. Enabling and emphasizing knowledge creation in the organization is one of the key topics for successful SPI (Mathiassen et al., 2002). Hence, experiences, good practices and other development knowledge need to be captured and evaluated in order to make them useful in SPI. In a software engineering community, software patterns have been adopted as an efficient way for knowledge capture, retention and retrieval (Coplien, 2000). Thus, this thesis studies patterns more closely as a tool to share and exploit design information, and further, as an enabler for organizational learning.

2.4 Patterns in software development

Software patterns are proven solutions to recurring design problems. They provide a tool to effectively communicate complex software engineering concepts and they can be used to record and foster reuse of proven solutions. Patterns capture essential parts of a design in a compact form, e.g. for a documentation of the software architecture (Beck, Crocker, Meszaros, Coplien, Dominick, Paulisch & Vlissides, 1996). Patterns are commonly listed in catalogues, either in pattern books or web pages, where patterns are often presented without explicit pattern relations. For example the book of Gamma et al. (1995) and Yahoo! pattern library[2] (see also Malone, Leacock & Wheeler, 2005) are both pattern catalogues. In pattern catalogues, patterns are mostly described in an informal way in natural language. This is an advantage of patterns, since understanding most part of a pattern does not require knowledge of notation semantics or of certain syntax (Hagen, 2002). Typical pattern presentation includes text that describes the pattern problem. Context and solution is then accompanied with graphical descriptions such as pictures, UML-diagrams or other notations. These notations further explain the desired design solution, its structure and consequences. Patterns can be used by programmers, designers and architects who are build-

[2] http://developer. yahoo.com/ypatterns

ing applications and want to improve either their understanding of architectural issues or their communication of them. Patterns are simultaneously seen to be precise enough to preserve the domain-specific knowledge but still "loose" enough to allow the systems' future reimplementation in related areas (Dixon, 2009; Olson, 1998).

Patterns as reusable components have gained research interest (e.g. Fach, 2001; Jacobson, Griss & Jonsson, 1997; Sodhi & Sodhi, 1998). Shortly put, patterns are reusable objects, but despite their similarities with software components, patterns are not components (Sodhi & Sodhi, 1998). Like components, patterns have interfaces, which enable them to communicate with other parts of the software and which hide the details inside the pattern. But, patterns are not independent; they can seldom work alone. Frequently, a number of patterns is needed to solve one design problem, and they interrelate with each other and work together like a network. Then again, context is present in patterns as well as in components. The three parts of patterns – namely problem, context and solution – reminds us that context is essential for a pattern to work. A pattern in the wrong context is called an anti-pattern (Buschmann et al., 1996). Patterns cannot be integrated into a system as such, but existing components can be arranged as described by a pattern. The result of using patterns, the software design, is not a pattern in itself. (Sametinger, 1997)

Software patterns and pattern books are commonly used both in the industry and in university curricula (Buschmann, Henney, & Schmidt 2007). Patterns bring systematization to the software development process, and they can be used in several ways to facilitate development work. Patterns can be used for example as

- means for handling complexity,
- analysis and description tool,
- means for communication and information exchange and
- means for learning and teaching.

2.4.1 Patterns for complexity handling

Complexity handling and the ability to make abstractions are central skills in software engineering (Kramer & Hazzan, 2006; Kramer, 2007; Wing, 2006). Patterns facilitate understanding of complex software compositions by promoting use of cognitive schemas (Robillard, 1999). Cognitive schemas are cognitive links to known terms. For example "text editor" is linked in our minds to a specific application, e.g. MS Word or emacs. In a software development context, the name of a pattern is linked to the specific problem and its solution. Consequently, this helps the developer to comprehend the complicated software structure and dynamics.

In abstraction, details are ignored at the expense of essential features. In computer science, abstraction is defined as a cognitive means that is used to overcome complexity in problem solving (Kramer & Hazzan, 2006). Presentation format of a software pattern containing details, graphical modeling and overview pictures, can greatly support abstraction. Patterns also support moving between different levels of abstraction. Hence, patterns can help to cope with the accidental complexity of software development and facilitate the learning and adoption of object oriented techniques (Astrachan, Berry, Cox, & Mitchener, 1998; Babar, Gorton & Jeffery, 2005). To explain this, the figure 5 gives an overview of Observer pattern. With the accompanying text, the picture allows the reader to grasp the main point: Defining a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

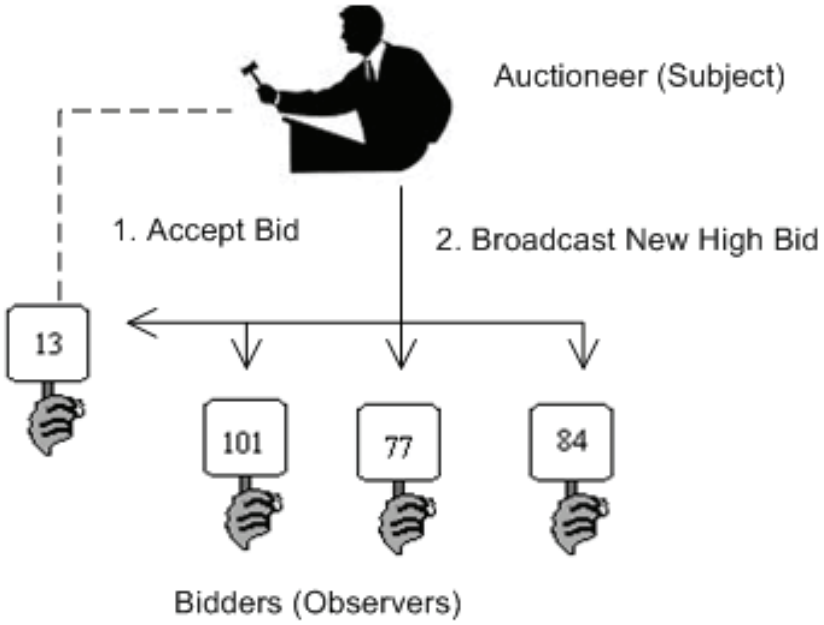

FIGURE 5 Observer pattern[3]

On this highest abstraction level no programming skills or UML knowledge is needed to understand the idea of the pattern. Then, a more detailed view on the same pattern is provided by the structure diagram, illustrated in figure 6. Accompanied with programming example3, the reader gets more advice on implementing the pattern.

[3] http://sourcemaking.com/de-sign_patterns/observer

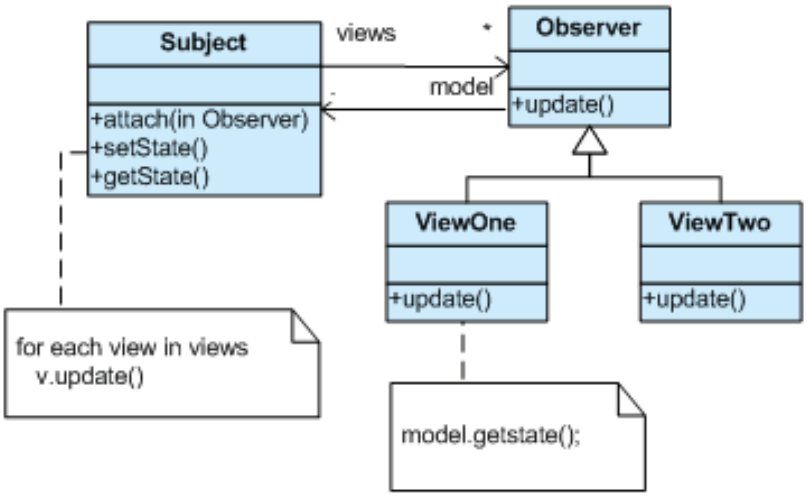

FIGURE 6 Structure of observer pattern[3]

2.4.2 Software analysis with patterns

A second use for patterns is in the analysis and description of software architecture. In this context, patterns are often divided into three groups: architectural patterns, design patterns and idioms (Buschmann, Meunier, Rohnert, Sommerlad & Stal, 1996). Each of the groups represents a different abstraction level. Architectural patterns are the most general level solutions, which express a fundamental structural organizational schema for software systems. They are templates for concrete software architectures. The architectural decisions are fundamental in nature and may have straight connection to the quality of the application and further to the time and costs needed (Babar et al., 2005). Design patterns are medium scale patterns, which describe relations between the subsystems in the software architecture. They facilitate the refining of the subsystems or components and hence, these patterns are interesting primarily to designers and programmers. The third group, idioms, deals with implementation of a particular design issue. Most of the idioms are language-specific. Idioms are smaller than components or patterns; they are straight coding examples and needed only by programmers. (Buschmann et al., 1996)

With a growing body of pattern literature, it has become obvious that there are numerous links between different patterns, and that patterns seldom appear in isolation (Marquardt, 2002). In addition, it is easier to understand and also to explain individual patterns when other patterns are cross-referenced

(Alexander, 1979; Porter & Clader, 2004). When patterns are organized in a systematic manner and their relations are made explicit, they form a pattern language (Alexander, 1979; Dixon, 2009). A pattern language explicitly relates one pattern to another in a hierarchical manner. Thus, a pattern language shows how the patterns build a larger entity, complementing each other in a way that they together solve a larger problem (Dixon, 2009). Pattern languages aim to provide holistic support to develop software for specific domains, such as e-commerce or communication middleware (Buschmann et al., 2007). Furthermore, a pattern language can be used to help the software engineers and domain experts to communicate their expertise to each other more explicitly and more precisely (Borchers, 2000).

The motivation behind pattern languages is also to strive for completeness within the tackled context or topic (Marquardt, 2002). Organizing patterns according to predefined principles allows them to form a living language, which can be used complementary to other languages in organization (Erickson, 2000; Dixon, 2009). When linked patterns and the linking principles are known, it may further facilitate selection of an appropriate pattern (Schmidt & Cleeland, 2000; Dixon, 2009). And like any other language, also pattern language changes and evolves according to its users' needs (Alexander, 1979; Dixon, 2009; Erickson, 2000; Schmidt & Cleeland, 2000). This allows the patterns themselves to change in compliance the changing environmental requirements, e.g. technologies. A further point in using a pattern language is, that it results in a design, in a sequence of patterns, not to a single solution in program code (Porter & Clader, 2004). Pattern languages support larger-scale reuse of software architecture and design than individual patterns (Buschmann et al., 2007). A pattern language can also be a part of organizational memory that handles design problems (Schmidt, Fayad, Johnson and guest editors, 1996; see also Booch, 2008b). A comprehensive pattern language is seen to provide the greatest payoff for pattern-based software development, even though developing one is challenging and time consuming (Schmidt et al., 1996).

Some researchers argue that having a common language may not facilitate, but will hinder communications in organizations (Erickson, 2000). The reasoning says that when using rich concepts of several disciplines, instead of limited vocabulary, it is more likely that a new and innovative design can be reached. However, a common language may encourage the participants to contribute to the design and to be heard in the discussions (Dixon, 2009). Common language can also lead the discussion into concrete issues and hence useless disputes may be avoided. For example, experts of different programming languages (e.g. Visual Basic, Java, C++) can share their design expertise without being caught in a "language wars" where mainly differences in syntax and semantics are discussed (Schmidt, 1995).

2.4.3 Tool for communication

In addition to the technical, another perspective on patterns is an organizational one. This perspective considers patterns as means to facilitate communication

and as a tool to learn. An organizational approach has been utilized, for example, in the study of designers' decision making (Wright, 2007) and in knowledge management (May & Taylor, 2003). Patterns have also been extended to cover organizational practices as process patterns (Coplien and Schmidt, 1995) and software inspection processes (Harjumaa, 2005). This thesis views patterns prevalently as tools to both collect and share expertise about design solutions, and for communication.

In software development, communication is required when sharing design information, in decision-making and in coordinating the design tasks (Chiu, 2002). Particularly large software projects often involve multiple people with different backgrounds (Korpela et al., 2002), who are possibly in different countries and have different mother tongues. Hence, communication of the software design becomes critical (Sarkkinen, 2006). In this communication, both visual representations, such as charts and prototypes, and verbal representations are important (Sarkkinen, 2006). A combination of these is needed for efficient collaboration (Hendry, 2004). Further, previous studies indicate that groups who define their terminology earlier in the design process, and groups who have a rich set of terms in use in their communication perform better than others in software projects (Subrahmanian, Monarch, Konda, Granger, Milliken, Westerberg & the N-DIM Group, 2003). Taken together, communication in software development is essential, time consuming and diversified in both topics and mediums. The different uses of patterns as communication tools and respective pattern characteristics are summarized in table 2. The leftmost column lists communication aspects that patterns can address; in the middle are the key characteristics of software patterns that address the identified communication aspects and in the right are examples of organizational communication needs where communication aspects and patterns can be present.

TABLE 2 Patterns as communication tools

Communication aspect	**Software pattern key characteristics**	**Examples of organizational need**
Common language	Pattern names and structure provide vocabulary for complex topics	Multi-site development, team communication
Knowledge capture	Packaged entity and structured presentation format	Innovation creation, message encoding
Boundary objects	Multiple perspectives on same topic	Stakeholder communication, communication in subcontracting
Documentation tool	Standard template	Project-based development, where developers change projects

In large companies in particular, communication can span teams, departments and products (Kettunen, 2003). Communication and coordination is required e.g. for enforcing software quality. The following figure 7 demonstrates patterns as communication tools with a commonly used strategy pattern. The

purpose of strategy pattern is to define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from the clients that use it. In the figure, the strategy pattern is exemplified with three different transportation methods. From a common language perspective, all the teams implementing different transportations would share similar view on the other parts of the system. In a subcontracting setting this could mean, that even if each transportation method is built by different supplier, the subcontractor does not need to reveal too many details.

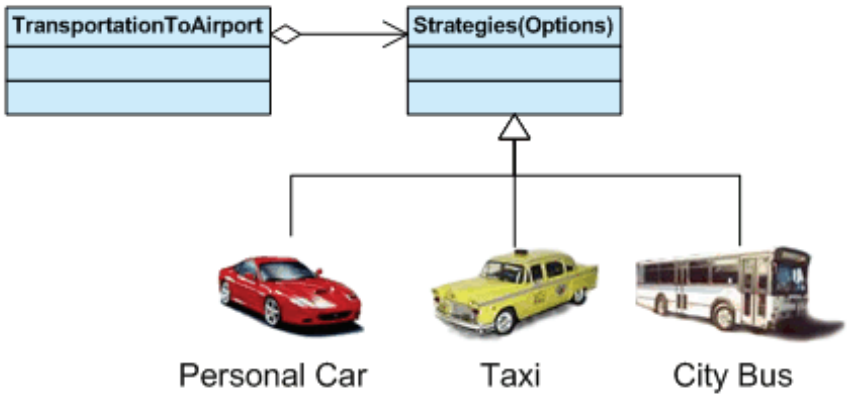

FIGURE 7 Strategy pattern: overview[4]

However, due to different perspectives on the topic at hand, cross-organizational communication can be extremely challenging (Korpela et al., 2002). Crossing the organizational boundaries can be facilitated by boundary objects, as noted earlier. Patterns and pattern languages could be used as boundary objects because of their flexibility. They are flexible enough to adapt to local needs and to the constraints of the groups using and adopting them, and yet they are robust enough to maintain a common identity across the groups (May & Taylor, 2003; Star & Griesemer, 1989). Considering software design, patterns and pattern languages can act as boundary objects since they grasp multiple perspectives on a single topic, containing information in different formats available for audiences with different skills (see Dixon, 2009). Considering boundary objects, the above picture demonstrates how a non-programmer can understand technical design. In other words, patterns can establish a shared language for the individuals, thus facilitating them to present their knowledge (compare Carlile, 2002). This kind of use can also push organizational boundaries (Lee, 2007), when people in different groups start to

[4] http://sourcemaking.com/design_patterns

adopt pattern vocabulary in their communication and hence can widen their own territory.

In addition to the correct vocabulary, communication as a rule requires two kinds of skills, namely the decoding and encoding of messages. Communication decoding means ability to listen, being attentive and responding quickly, while communication encoding is an ability to express one's ideas clearly, have a good command of the language, and be easily understood (Monge, Bachman, Dillard & Eisenberg, 1982). For software developers, communication about complex issues is crucial (Korkala & Abrahamsson, 2007), but not always simple. Patterns can facilitate developers' communication since they capture a design solution to a packaged knowledge entity (Beck et al., 1996; Babar et al., 2005; Cline, 1996). Using design patterns as a means of communication, the issues discussed will become easier to understand and easier to refer. Patterns can also provide a list of things to look for, e.g. during a review (Cline, 1996). This kind of common vocabulary can facilitate communication both between team members and between different teams (Borchers, 2000; Schmidt, 1995).

However, software patterns are not merely for communicating and finding design solutions; they are also for balancing the system under examination (Dixon, 2009). As a standard format, patterns provide a concrete means to specify and learn about the differences and dependencies between design alternatives. Consequences of each pattern are visible in the presentation, which allows the developers to negotiate the candidate solution and its trade-offs. Also the context of the patterns is explicitly described, which has been noted to be crucial for boundary objects (Lee, 2007). In summary, these characteristics indicate that patterns indeed can work well as boundary objects in software design (Carlile, 2002), assuming that the negotiating parties are familiar with patterns.

Furthermore, patterns are a means to document the knowledge that has been acquired, presented and used by different developers during several projects and maybe in different organizations. Using a common pattern template in documentation transfers the implicit knowledge to explicit and enables the expertise to be shared and preserved (Schmidt, 1995). In addition, when patterns have been used in the software design, their documentation enables the maintainer or other reader of the code to easily understand the design tradeoffs that have been adopted in the design (Babar et al., 2005; Cline, 1996;). Further, new ideas about appropriate solutions can be made explicit by using specific kinds of documentation. The documentation can help the decision makers and end-users of the system to discuss the features in advance in the course of design process and the developers get valuable feedback of their efforts in previous phases. This gives the developers a chance to benefit from the experience of others and to avoid mistakes in the later phases of the development process. (Herrmann, Hoffmann, Jahnke, Kienle, Kunau, Loser & Menold, 2003; Schmidt, 1995)

2.4.4 Tool for learning

In addition to communication, software patterns can also serve many learning purposes. Their structure can benefit both the learner and the teacher in expressing and grasping the information and turning it into knowledge (Buschmann et al., 2007). In this context, patterns can be used

- as information sources,
- to create collective understanding,
- for training new developers and
- in teaching object-oriented thinking.

Generally developers use pattern catalogues as information sources (e.g. Booch, 2007; Gamma et al., 1995). Often, particularly trustworthy information sources are valued above others; hence information source quality becomes an important aspect to evaluate (Herzum, 2002). Furthermore, many developers favor personal and interpersonal information sources and internally published technical reports over externally published documents (Tenopir & King, 2004). Thus, software patterns, particularly in-house patterns, can serve both as an information source and as a communication mediator in oral communication, for both their quality as a design solution and their reliability can be rated (Agerbo & Cornils, 1998) by the software development community in general and also by the particular developers.

However, books and other information sources are not enough to do brilliant software design. Learning at both individual and organizational levels involves the transformation of data, i.e. un-interpreted information, into knowledge, that is interpreted information (Popper & Lipshitz, 2000). Patterns with complete structures that explain all problems, contexts and solutions, can serve as a tool in this transformation. Furthermore, creating innovative and high-quality design solutions requires collaborative developers who value collective understanding as a development practice (Wirfs-Brock, 2009). Building such collective understanding is not always easy, but it can be facilitated by means of cognitive elaboration and model building (Zhong & Majchrzak, 2004). Thus in practice, learning can be fostered by mediating the message with many different means: by using pictures, graphs, text, concrete models and possibly other tools to explain the same contents (Sarkkinen, 2006). This allows the learners to gain a multi-sided view of the topic and encourages the creation of collective understanding. Since patterns contain information with many presentation formats, they can be valuable in fostering learning within various roles in software development.

In a training context, software patterns have been considered as means for mentoring a novice programmer (Beck et al., 1996). Patterns help less experienced designers apprehend the collective wisdom of other designers in a way that can be immediately used (Buschmann et al., 2007). Furthermore, experienced developers already have solutions to many recurring design problems,

but a pattern catalogue can help a less experienced programmer to learn about those techniques. For example, the strategy pattern illustrated in figure 8 teaches a fundamental principle of object-oriented thinking, namely the "open-closed principle". According to this principle, the details of the interface are encapsulated into the base class, while implementation details are hidden into derived classes. This structure lets the algorithms vary independently from clients that use them. Presenting the lesson with several graphical pictures and explanatory text lets the novel programmer to understand the principle and its importance. Programming code example then allows the adoption of the principle directly to one's work.

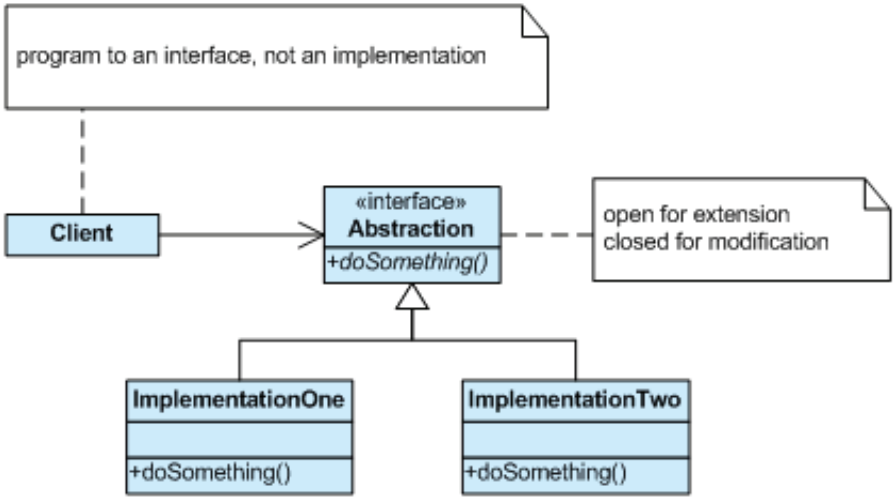

FIGURE 8 Strategy pattern: hide details[5]

In addition, a pattern catalogue can help both experienced and novice developers to recognize situations where reuse could or should occur (Beck et al., 1996). Also, when recruiting new members to the development team, patterns can be used to teach the product design to the new arrivals (Cline, 1996). For those who already are familiar with the pattern techniques, a pattern catalogue can help to communicate them to others (e.g. Fowler, 2003). Thus, for novice programmers, it can help greatly to have design patterns present, for in traditional education methods much time is spent learning the language syntax, while overall picture of what is essential and how the pieces fit together is lost (Proulx, 2000). Further, patterns can be used as a part of more complicated software that could not be implemented completely by the trainees themselves. Also, when trainees have experience only on structured software engineering

[5] http://sourcemaking.com/design_patterns

paradigm, design patterns can facilitate transition to object-oriented programming and way of thinking (Astrachan et al., 1998).

However, it is worth noticing that patterns are not learnt merely by reading a pattern catalogue, but learning patterns requires their active implementation. This enables the learners to become familiar with the pattern way of thinking and to master the pattern's use. However, the problem domain where patterns are implemented must be familiar to the learners beforehand. When the example problems are kept simple enough, the learners can better concentrate on each pattern's purpose rather than trying to make all the patterns fit into one single example. (Goldfedder & Rising, 1996)

2.5 Bridging the gap between SPI, organizational learning and software patterns

In the previous chapters, relevant research on SPI, organizational learning and software patterns has been presented. It was noted that systematic organizational learning is a prerequisite for SPI. Further it was pointed out that knowledge management provides central improvement targets in order to increase process efficiency and hence also product quality. Patterns were presented as a knowledge management tool that can be used to capture and share design knowledge.

Current research on these topics is dispersed, although organizational learning is widely studied in connection to software process improvement (e.g. Heikkilä, 2009; Seigerroth & Lind, 2006). Patterns have been applied in process modeling (e.g. Gschwind, Koehler &Wong, 2008), in groupware development (Schummer & Lukosch, 2007) and they have garnered interest as a software development tool (e.g. Buschmann et al. 2007; Dixon, 2009; Soundarajan, Hallstrom, Shu & Delibas, 2008). However, research on software patterns and organizational topics seems to be more limited (Booch, 2008; May & Taylor, 2003; Vesiluoma, 2009). This dissertation aims to bridge the gap between these research topics and contribute to knowledge on methods for software improvement.

Software patterns and continuous learning are key issues also in agile development (Buschmann et al., 2007; Salo & Abrahamsson, 2008;). Hence, the importance of this research is further emphasized by the mushrooming of various agile methods in both academia and software industry (e.g. Dybå & Dingsoyr, 2008; Salo & Abrahamsson, 2008). In addition, the increasing use of open source software and other technological advancements bringing challenges to software development (Buschmann et al. 2007) call for this research.

3 RESEARCH PROBLEM AND METHODOLOGY

The following subsections outline the research problem and methodology. First the research problem is presented. After that the research methodology and data collection methods are described. Finally, the relationships of the research topics and the included articles are depicted.

3.1 Research problem

Software processes are complex by nature, partly because of rapidly evolving technological environments and partly because of the intangibility of the produced artifact. At the same time software organizations face heavy competition and thus need constantly reassess their working procedures. SPI provides models and methods for process assessment and process tuning. To identify, plan and embed the improvements, responsive and active individuals are needed. In addition, organizational learning is required in order to make the improved practices last. Furthermore, as indicated in the previous sections, organizational learning does not happen automatically. Instead, to enable the individuals to learn in the hectic software development environment, organizational support for such learning is a necessity. Since organizational performance, and thus SPI, often depends on ability to turn knowledge into effective action (Alavi & Leidner, 2001), efficient knowledge management is also required. Therefore knowledge sharing is also essential. Consequently, in this thesis knowledge sharing in software development is studied with software patterns. This thesis examines software patterns as tool to manage software design knowledge and particularly as tool for communication and learning. In summary, this thesis aims at:

- providing an analysis of needed organizational learning in software process improvement
- providing guidelines for software patterns' use as a learning enabler.

Patterns have existed in the software community for years. However, both research and industrial practices have concentrated on technical pattern topics, i.e. on improving software design, instead of the necessary learning effects (e.g. Buschmann et al., 2007; Fowler, 2003; Gamma et al., 1995; Schmidt, Stal, Rohnert & Buschmann, 2000; Yacoub & Ammar, 2004;). In order to better exploit patterns' full capabilities, further investigation on patterns and particularly their properties as a communication and learning tool is highlighted. Hence, relating this to the tentative research questions, the second research question is reformulated to better cover the versatility of patterns. As a result, the two research questions of this thesis are

- How can organizational learning be supported in software organization?
- How can software patterns serve as a tool for knowledge sharing in software development?

In this dissertation the above research questions are analyzed with five articles. Figure 9 depicts the articles with the central research concepts. In the first article, the need for organizational support to enable learning in software organizations is investigated. This study forms the basis for the following articles. Improvements for both knowledge sharing and organizational learning are then suggested in the second article. In the third paper, research focus is shifted to software patterns in organizational communication. Patterns as a part of organizational memory are then examined in the fourth paper. Finally, the fifth article approaches software process improvement and the required knowledge management from inter-organizational perspective.

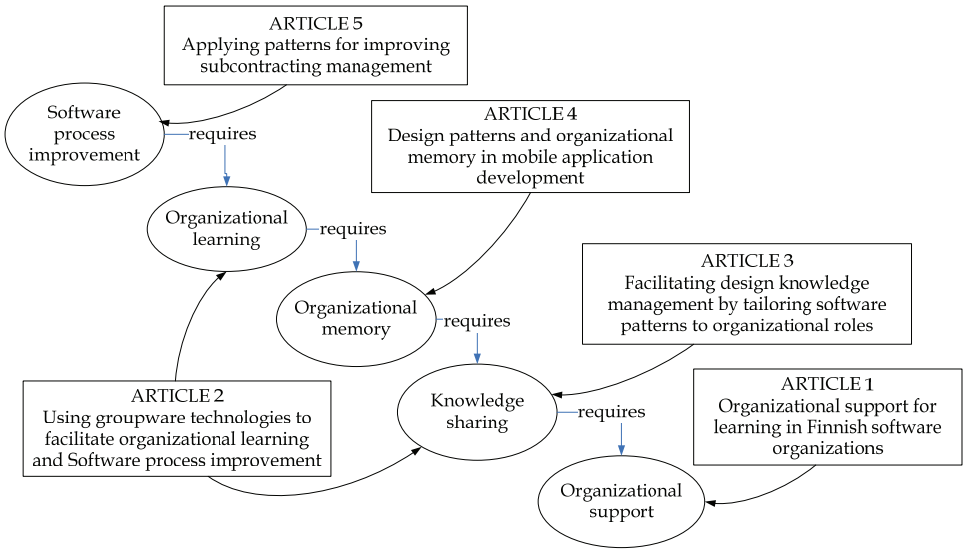

FIGURE 9 Research constructions with related articles

As a main result, this thesis provides ideas and guidelines on how process improvement can be enabled through the promotion of organizational learning in software companies. As discussed in the introduction, this study uses a knowledge codification approach by incorporating and assimilating previous knowledge and further by creating new knowledge in the form of a dissertation. As a result it is noted, for example, that an organization can support individuals' learning by having structured practices for receiving feedback and for setting individuals' goals. Other results indicate that creating a supportive climate for process improvements can be facilitated by using groupware and that the communication within a team, such as goal setting, can be improved with software patterns. Development of organizational memory and further knowledge accumulation to an organizational level can be promoted by regarding the software patterns' use as a dynamic process. Finally, research results indicate, that organizational learning in an inter-organizational setting, particularly in subcontracting, can be structured by using software patterns.

3.2 Research approach and methodology

Generally, research method selection should be based on the research setting and the problem. Not only the researcher's experiences and views, but also the discipline, and the location of the research have an influence on the approach. For example, information systems (IS) and software engineering (SE) can be seen to study similar phenomena with different approaches. SE approaches software development from an engineering perspective, i.e. technical and mathematical, while IS brings in managerial and social aspects. This dissertation exploits concepts from both disciplines, since the central research topic, software patterns, is traditionally a research object for SE research. In this study however, the concepts mainly studied in the IS field, namely process improvement, organizational learning and knowledge management, are used to describe software patterns' use.

This thesis approaches software process improvement and organizational learning by describing and explaining the current company practices in software development. Organizational perspective was selected to ensure that the results are directly applicable in SPI initiatives. However, since SPI often comprises small steps in individual and team practices, which are then distributed and embedded in company level, this study also needed to study individual and group levels of organizational learning. These were particularly necessary in accomplishing a holistic view of the topic and to keep the study in practical, applicable level. The selected organizational perspective further affected on the selection of the background theory of the research. Organizational theories on learning and knowledge management were used to define the research focus and they were used to interpret the collected data.

Considering data collection, both quantitative and qualitative methods were included. This was to gain both an overview and a deep insight into the

topic. Quantitative methods were chosen to provide the overview and to serve as a motivation for the study. With qualitative methods the research aimed to deepen the understanding of the revealed issues and to allow further analysis of the research topics. The relation of the research methods to the research objectives is illustrated in figure 10.

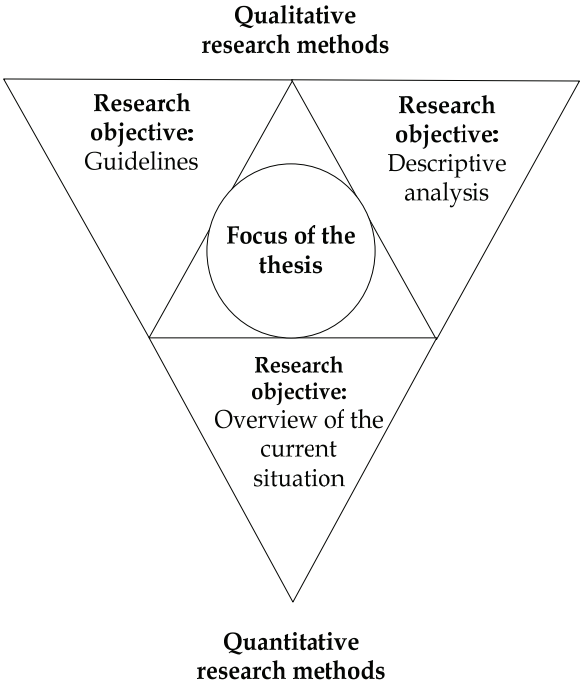

FIGURE 10 Research objectives in relation to the research methods

To create motivation and to gain understanding of the of the research area, an overview of organizational support for individuals' learning was needed. Survey research was chosen to serve best as a method, for the topic was well defined beforehand and the aim was to reach large number of respondents (Järvinen & Järvinen, 2000). Case studies in general are suitable to deepen the researcher's understanding of the specific issues (Stake, 2000). In this thesis, multiple case studies were selected to provide deep but varied information on the daily practices in the organizations. To differentiate between multiple case studies and survey sampling, the *replication logic* is of utmost importance (Yin, 2003). Following this logic, in this research each company is handled as its own case, and thus the particular research topics related to the particular cases are defined. However, the business field of the companies is common to all the case companies, namely the mobile phone business, which thus forms a congruence to the research. The aim of this research approach is to enable a varied view to

the actual research problem and simultaneously to have a positive impact on the generalization of the results.

Data collection in the companies was carried out through interviews and document analysis. These methods were selected to allow interaction with the people and thus served as a gateway to the actual organizational practices. In case work reflection of the cases is crucial and it can be conducted in several ways, as Stake (2000) suggests. In this thesis the reflection was carried out by analyzing previous research, the theory of organizational memory, and the model of software process improvement (CMMI). These were then related to the organizational environments of the case companies. This approach was selected to produce generalized descriptions for alternative solutions and improvements of the existing company practices.

Research validity refers to the degree of accuracy with which the research reflects or assesses the specific research object. According to Yin (2003) the four important aspects of validity are construct validity, internal and external validity, and reliability. Construct validity refers to the degree to which the research design is able to capture the reality. In this research construct validity is central, particularly in describing the current company practices, for absent construct validity can give a totally false starting point for the rest of the research; hence, the suggested improvements would also be ineffective. In this research, construct validity is addressed by having multiple companies as case organizations. Further, the drafts of case study reports were distributed to the key interviewees for comments, as suggested by Yin (2003).

External validity refers to the potential for generalizing the research results, while internal validity referring to causal relationships is not relevant to descriptive studies (Yin, 2003). In this research, external validity is important since the case companies were all Finnish companies developing mobile phone applications. Hence, external validity facilitates the general application of the results to other software companies in other countries. External validity of this research is addressed by using varying research methods, i.e. method triangulation, as suggested by Stake (2000). The fourth component of research validity, reliability, refers to repeatability of the research operations. In this research it is merely addressed separately in each of the included articles by describing the used research processes and data collection methods.

3.3 Relationship of the included articles

This thesis consists of five articles that approach the topic of software process improvement and organizational learning from different viewpoints. Nonaka's (1994) theory of knowledge creation utilizes four levels of knowledge creation entities, i.e. individual, group, organizational and inter-organizational level. New knowledge is created on these levels and happens on interactions between the levels. The five articles constituting this thesis touch all the levels, as illu-

strated in figure 11. This figure associates the research topics with the included articles and finally also the research results.

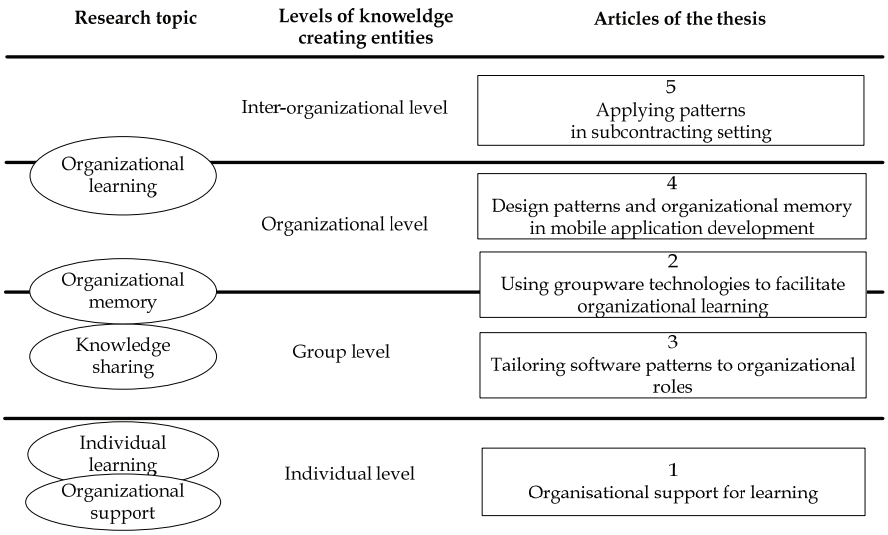

FIGURE 11 Articles and topics of this thesis in the levels of knowledge creating entities

The individual level of knowledge creation entities is researched in the first article by studying the individuals' experiences on learning and organizational support for learning. The second article touches the group level by having team communication as one of the focus areas, but concentrates merely on the organizational level by studying the creation of supportive organizational environment. The third article concentrates on team level by studying patterns as a tool in communication between roles. The fourth article then focuses again on the organizational level, by researching software patterns in relation to knowledge accumulation and organizational memory. Finally, the fifth article touches upon the inter-organizational level, when pattern benefits are studied in subcontracting relationships. In summary, the five articles constitute research that analyses and evaluates organizational learning practices in SPI. The research proceeds from individuals to teams and organizations and includes findings from both literature and industry.

4 OVERVIEW OF THE PAPERS

This chapter briefly summarizes the five articles included in this thesis. In the case of co-authored papers, the authors' contribution to each paper is also described. The contributions of each article are described in greater detail in the last section of this dissertation.

Paper 1: How Organizations Support Learning in Practice: A Survey on Finnish Software Organizations

The first article approaches software process improvement and design knowledge sharing from the point of view of an individual's learning. In order that processes may be improved, the individuals first need to learn new ways of working and new skills to perform their work more effectively. In Nonaka's knowledge creation theory, the internalization process describes how individuals must first internalize and learn before he can share his skills to his peers. To enable the individuals to learn, an encouraging and supportive organizational environment is required. In the first article, learning enablers (van Solingen, Berghout & Trienekens, 2000) are used as a measurement for the organizational support.

The research was conducted as an internet survey, targeted at people working in software development or information system development. The final data comprised 195 respondents. The exact research areas included in the survey included the climate of openness, scanning for knowledge, getting information on context and the current state of the system, team learning, modeling of the system under control, possibilities for control, involved leadership, explicit goal definition and monitoring the performance gap. Based on these learning enablers and their adoption in the organizations, the article describes the organizational support for learning in Finnish software organizations. As a result it is concluded that, in particular, the goal setting for the work and feedback practices need improvement to enable the software engineers' learning.

The paper was written collaboratively with Samuli Pekkola. Anne Pirinen participated in research planning. Riikka Ahlgren was responsible for gathering the data and analysis of the results.

Paper 2: Using Groupware to Facilitate Organizational Learning And Software Process Improvement - A Case Study

The second article approaches software process improvement and organizational learning from a communications perspective. As highlighted in article 1, communication is essential in creating a supportive organizational environment suitable for learning and process improvements. The case study investigated how groupware can support this communication. In the article communication quality within a team is described with four attributes: communication openness, formalisation, frequency and structure (Hoegland & Gemuenden, 2001). These attributes contribute to efficient information exchange by different means. Communication openness promotes team integration and thus serves for collective learning; changing formality allows concentration on the contents and not on the communication itself; frequency lessens possibilities for misunderstanding; and appropriate communication structures improve team members' equality and further facilitates the co-operation.

In the case study, several groupware technologies and their use were researched. The results were analyzed from both individual and organizational perspectives. The results indicate that various groupware tools can be used for sharing daily development information, but they can also serve as information channels on organizational topics and thus facilitate organizational support for both learning and software process improvement. In one case, organization individual learning was supported by groupware. In particular, chat software allows developers to freely discuss open issues and solve possible problems together, without formal meetings. In another case, communication of organizational visions was supported through blogs and email. This was further seen to support the creation of an organizational policy. This communication was found to improve the software development when developers were allowed to concentrate on their own tasks without need to ponder managerial decisions. The study was planned and conducted together with Mirja Pulkkinen. Results were analysed by Riikka Ahlgren and the theoretical implications were provided by Eleni Berki and Mirja Pulkkinen. The paper was written collaboratively by all authors.

Paper 3: Facilitating design knowledge management by tailoring software patterns to organizational roles

The third article elaborates the requirements and problems of knowledge sharing in software development. The article concentrates on design knowledge management between different roles within a software development team. Differing skills and backgrounds often cause misunderstandings and can thus endanger work estimates and goal setting in the team. The article focuses on using software patterns as a communication tool between different roles. The research was motivated by a case study that was conducted in a software organization interested in promoting patterns. The results of the study indicate that there is often a communications gap between technical and business staff. This can make goal setting difficult in software development teams, if the project managers do not have enough knowledge on technical aspects of the software engineers' work. The communication problem raised by the case study was then addressed by creating a theoretical model, where patterns are tailored according to different roles. The structure of patterns was analysed and different pattern parts and their properties were studied. The results indicate that patterns can support structured communication between different roles. Thus, both technical and business-oriented staff can benefit from patterns as means of communication.

Paper 4: Design Patterns and Organizational Memory in Mobile Application Development

The fourth article focuses on the problem of preserving and accumulating knowledge in mobile application development. Complicated and rapidly evolving technological environment highlights the need to accumulate individuals' knowledge at the organizational level. To gain the benefits of existing design solutions, and to be able to create new innovative solutions, the accumulation of knowledge is essential. The accumulation however is not obvious, since the knowledge lies in individuals' heads and can leave the company when the people do. To address this problem the article concentrates on software patterns in relation to the theory of organizational memory.

Organizational memory is defined as a dynamic process comprising four processes, namely knowledge acquisition, retention, maintenance and retrieval, which are in constant interaction with the organization's structures, practices and tools (Stein, 1995). In the article, the utilisation of design patterns is analyzed in comparison to these four phases. Based on theoretical research analysis, it is suggested that the patterns' properties allow them to be used as knowledge entities constituting organizational memory. As a result a framework is built, where pattern acquisition, retention, maintenance and retrieval are depicted

within the organizational structure, practices and tools. The developed framework provides a new view to software patterns and it can be used to facilitate a pattern's adoption to daily development work. This article was co-authored with Jouni Markkula. Research planning and conduction was carried out by Riikka Ahlgren.

Paper 5: Applying Patterns for Improving Subcontracting Management

The fifth article of this thesis considers organizational learning and knowledge sharing at the inter-organizational level. The motivation for the article is raised by the changing business environment in software development. Subcontracting and subcontracting management is used in software business to quickly and efficiently respond to business changes. Thus, it has become one of the key success factors for software companies (Herbsleb, Mockus, Finholt & Grinter, 2001). However, several problems have been encountered in such collaborations. The problems in software subcontracting can range from the contracting itself, requirements engineering, project management, to overall quality of the subcontracted component and foremost to the communication that is needed in the different phases of the subcontracting process (Assmann and Punter, 2004; Paasivaara, 2003; Smite, 2005).

A well-known problem in software subcontracting is defining the work contents at an adequate level. In this article, software patterns are analyzed through specific CMMI goals of establishing and satisfying supplier agreements. The analysis concludes that patterns' key property in subcontracting management is to deliver information about the system design without revealing the implementation details. As a result of the article it is suggested, that software patterns can be used as a tool in process improvements related to subcontracting management. Particular benefits are suggested in the practices of selecting suppliers and in establishing supplier agreements. This article was co-authored with Jari Penttilä and Jouni Markkula. Research was planned by Riikka Ahlgren and Jouni Markkula, and the final article was written collaboratively by all authors.

5 CONCLUSIONS

Software process improvement is largely about standards and models. The research and case studies both suggest that improving knowledge management can provide major improvements for the development process. However, case studies indicate that in software organizations SPI is small things on a small scale. For example, in these organizations SPI means minor changes to document templates or sharing information with communication tools. Only when these small steps succeed can the scale be enlarged. Hence, to succeed in the first place, improvement initiatives first need skilled people: skilled at performing development tasks, skilled in learning and questioning the current practices and skilled in sharing their competence. The aim of this thesis was to give some practical guidelines on how such small process improvements could be supported. This aim was approached from an organizational learning perspective, as it is a prerequisite for the process improvement. Furthermore, the diversity of knowledge sharing in software organization is highlighted, as it is crucial both in the daily design work and also in the actions of management. Hence, knowledge sharing is essential for the whole software organization to learn. The particular research questions of this research were "*How can organizational learning be supported in software organization?*" and "*How can software patterns serve as a tool for knowledge sharing in software development?*" The following sections offer findings on these research questions.

5.1 Supporting organizational learning in software organization

Organizational learning is a necessity for a software company to successfully improve its processes. Both the literature review and case studies indicate that even though organizational learning can happen unintentionally (Bennet & Bennet, 2008; Huysman, 2000), it is more efficient and can be better exploited when it is intentional (e.g. Huysman, 2000; Yang, 2007). Organizational learning is enabled only if individuals are capable of learning and actually understand-

ing the learned topics. Furthermore, they will need to share their knowledge with their peers. The shared knowledge must then be embedded within the organizational processes and practices. Only then can the employees decode and adopt this knowledge in their daily development tasks. And only then can the software practices and processes improve.

Regarding the first research question of this dissertation "*How can organizational learning be supported in software organization?",* support for organizational learning was presented in all five articles. The results of the survey questionnaire in the first article indicate that organizational support is often lacking from software organizations. It was noted that there are communication problems between management and developers, and that an open climate is crucial when fostering learning. Communication problems between management and developers were also identified in the third paper. Tailoring a software pattern presentation format to business roles was suggested to increase managers' understanding of development work. This can bring two perspectives, namely business and technological, closer to each other and hence, can foster creation of a shared vision and supportive organizational climate.

An open organizational climate can also be fostered with technologies. Groupware tools and their benefits were analyzed in the case study reported in the second article. It was suggested that managerial communication and attitude towards communication tools can have a great effect on both experienced climate and on developers' learning at work. In practice, management's supportive attitude and open communication on organizational level can foster developers' mutual communication, which further encourages learning and can even enable innovation. The fourth article provided other examples of required organizational support. In that study, the concepts of organizational memory were connected to the use of software patterns. It was noted that in an organization there are several organizational factors that can have effect on patterns' use and on effective knowledge sharing. These factors include: allocated time for pattern learning, general quality attitude and quality awareness, and organizational structures such as experts' meetings. The fifth article suggests that organizational support for learning is also needed in inter-organizational settings. However, in the subcontracting setting that was studied, the support is needed merely from the client for the subcontractor. It was found that client organization can support their subcontractors' learning and hence their process improvement activities by providing guidance in adopting new working methods.

5.2 Software patterns as tool for knowledge sharing

The second research question "*How can software patterns serve as a tool for knowledge sharing in software development?"* was examined by analyzing pattern structures, properties, and their use and potential uses within different knowledge sharing settings in software development. Sharing of various types of knowledge, particularly design knowledge, is a constant activity during software de-

velopment. Knowledge exists in different formats, is situated in different locations and it also varies by nature. The knowledge sharing purposes also vary: a need for both detailed analysis and complexity handling is constant; new solutions to old and new problems must be searched and adopted; design decisions need to be made. Work must be divided between developers, and novices need expert opinions to help them out. Furthermore, communication requires various skills. Recognized skills include the ability to listen and a good command of language. Consequently, communication takes increasing amounts of time for developers and other team members.

Patterns were found to possess several characteristics that support communication in software engineering. Patterns have a structured, yet flexible template that enables effective design knowledge capture, communication and retention. The template includes information on several abstraction levels, which support both software analysis and complexity handling and supplement developers' communication of the issue. Furthermore, different abstraction levels and various presentation formats in a pattern template facilitate communication between communities of practice. Hence, patterns can work as boundary objects, facilitating understanding between groups with different backgrounds. In this regard, patterns' flexibility to different audiences is particularly valuable, since many design processes depend on the participants interpreting boundary objects, not in the same way but in compatible ways (Eckert & Boujout, 2003).

In addition, patterns can form a common vocabulary for software engineers of different domains and with different backgrounds. In particular, a pattern's name can serve as a reference to be used in communication. This enables discussion to flow smoothly, when the participants need not explain every little detail, but can refer to the pattern simply by its name. This is essential, especially in a project organization, where developers can switch teams in few months' intervals, and hence need to communicate software design with new people. In summary, patterns seem to have many beneficial characteristics, which encourage their systematic use in software development. In particular, pattern structure and presentation format enable them to serve as tools for knowledge sharing in software development. Connected to explicit SPI goals, patterns' benefits can be escalated even in inter-organizational use (e.g. subcontracting). In that context, patterns can be exploited in making explicit work agreements, in controlling and monitoring project and possibly also in protecting strategic business information.

5.3 Closing remarks

This dissertation has studied knowledge sharing on all the levels of knowledge creation entities. In the following table 3 these levels and research findings are summarized. As stated in chapter 3.3, new knowledge is created on these levels and in interactions between the levels. Hence, this makes them important points

of analysis in this context. In the first column on the left, the levels of knowledge creation are listed. In the middle, the organizational support for learning relevant at these levels, i.e. findings from the first research question, are characterized. In the column on the right, the second research question is summarized with the pattern characteristics and uses in knowledge sharing on each level.

TABLE 3 Research findings in the levels of knowledge creation entities

Level of knowledge creation entity	**Organizational support for learning**	**Software patterns in knowledge sharing**
Inter-organizational level	Adopting work methods	Standardized knowledge, structured template
Organizational level	Organizational structures, quality attitude and awareness	Knowledge entities
Group level	Communication tools, organizational structures	Boundary object, common vocabulary
Individual level	Open climate, feedback, clear objectives	Structured template

Concerning the organizational support for learning at the individual level of knowledge creation, it was concluded that open climate, feedback and clear objectives are the three most essential factors that enable developers' learning. The open climate can best be created on a group level or at an organizational level of knowledge creation, since it requires communication tools, organizational structures and general quality attitude. At the inter-organizational level organizational support for learning was studied in a subcontracting setting. It was found that new knowledge can be created in many phases of the subcontracting relationship. The client can support their subcontractor's learning for example by providing support in adoption of new work methods.

To sum up the findings of the second research question concerning patterns in knowledge sharing, it was found that software patterns can be used in different ways at the different level of knowledge creation entities. At the individual level, the greatest benefit was the structured template that can help the developer to express and classify its thinking and knowledge on complex issues. At a group level, software patterns form a common vocabulary that facilitates communication on design issues in a development team or between teams. Furthermore, software patterns can be used as a boundary object between groups from different fields, as their presentation format supports abstraction and understanding by different audiences. This was demonstrated in figures 5 and 6 of this thesis. At the organizational level of knowledge creation, patterns represent pieces of organizational memory that can be acquired, stored, maintained and retrieved. This enables the captured knowledge to accumulate and be accessible for the whole organization. At the inter-organizational level of knowledge creation, patterns represent standardized pieces of design knowledge, which allows

them to be used as work objects or quality measures. For example patterns “Remote proxy” and “Synchronization”, presented in paper 3, can demonstrate this use.

6 CONTRIBUTIONS AND LIMITATIONS

Competitiveness in the software business largely depends on high-quality products and efficient development processes. This brings process improvement into the very heart of a software organization's competency kit. The main contribution of this thesis is to analyze software process improvement in terms of organizational learning and knowledge management. It cannot be overstated that software process improvement means *learning*. It is learning at several levels of the organization, all of which are equally important for the SPI to succeed. This thesis highlights the need for communication in knowledge management, learning and in software process improvement. Communication is an essential part of all development phases and it is a central task for all the roles participating in development processes. For software organizations this study suggests concrete means to improve communication during development, and gives practical ideas on improving design knowledge management. The main contributions of this dissertation are presented below. Research limitations are then noted and finally, further research topics are pondered.

6.1 Contributions

This thesis suggests that patterns can be small steps on a small scale, the ones that are necessary for the individuals, teams and organizations to learn, and the ones that are needed for successful SPI. Hence, this dissertation provides a useful and an interesting perspective to SPI, where knowledge sharing plays a critical role. In the study, the software patterns were investigated as communication tools in software development and their use was connected to organizational learning. From an academic perspective, this view brings the two approaches, software engineering and knowledge management, closer to each other and thus provides new and fruitful insights on both perspectives. In particular, the study on software patterns as part of organizational memory dem-

onstrated how the theories of two different fields can be exploited in order to gain a better understanding on both topics.

In this study, software patterns were found to have capabilities for versatile uses in varying communication settings. Pattern characteristics were combined with organizational communication needs that hence provide a practical listing of their use-cases. These use-cases can be adopted in both academic research and in industrial settings. Furthermore, it was suggested that patterns can work as boundary objects that can facilitate learning and communication in software development. This can further foster improvement of software processes. Considering the commonality and significance of communication problems in software development, these contributions are truly important.

Further contributions were achieved by expanding the patterns' use in an inter-organizational context, namely subcontracting. Patterns were seen to provide control over a supplier's development process. This idea can be escalated even to in-house projects, where integrations are crucial between a common platform and various applications. This kind of development setting is similar to subcontracting, where one team builds the platform, i.e. basic structures and services, on top of which other teams then build their applications. In both types of projects well-documented, flexible and high-quality design solutions together with fluent communication and coordination between the teams are crucial for success. In both cases, whether it is a subcontracting or in-house setting, patterns can provide models for both technical design and communication facilitation.

The contributions of this thesis are valuable both in academia and industry. The areas for improvement span general business competency to specified software development practices. At the most general level, the business competitiveness of a software organization can be increased by acknowledging the learning aspects of process improvement. Such a systematic development of selected organizational skills can be a great advantage in a rapidly evolving software business environment. This thesis suggests knowledge management and, in particular, the sharing of design knowledge are central skills to develop in a software organization. These skills are required especially in performing specified development methods and practices in an efficient manner.

In addition to systematic learning of organizational skills, further value to a software organization can be added by efficient software development methods and practices. In the presented research, the efficiency of methods and practices is particularly examined through software patterns. The research indicates that the practices that can be improved with patterns of, for example, developer training, project coordination and product documentation. Considering developer training, patterns were suggested as information sources or as quality standards from which to learn. Project coordination in turn, both for in-house projects and for subcontracting, can be improved through the increased visibility and control that patterns provide to the development. Further, product documentation can be improved by referring to the patterns used, or by using a pattern as a documentation template.

Besides business benefits gained through learning and efficient practices, a central contribution of this thesis lays on the fact that innovation is born in interactions (Alavi & Leidner, 2001; Calantone et al., 2002; Nonaka & Takeuchi, 1995). Hence, fostering practices and tools that enable interactions between people and teams can also spur innovation. This is particularly important in the software business, where development teams are an important source of innovation (Conradi & Fuggetta, 2002) and where innovations have a crucial role in the business success. Hence, this research further highlights the value of patterns as communication facilitators, and particularly emphasizes their possibilities as boundary objects. For academia, this perspective provides an encouraging topic for further empirical studies.

Taken together, software organizations can exploit these results in several ways when planning their future SPI initiatives. By identifying pattern characteristics and applying them to specific development practices, this thesis gives software organizations ideas for intensifying and rationalizing their pattern use. Furthermore, software organizations can use the results in order to improve software processes and in preserving these improvements. In academia, this research builds on knowledge of SPI, organizational learning and software patterns. SPI was described in organizational terms, where communication and knowledge sharing is in central role. Particularly, knowledge of software patterns is increased with the perspective taken in this dissertation, as software patterns were suggested as a tool to embed the learning into organizational use.

6.2 Limitations

Certain limitations need to be considered when assessing the results of this research. First, the selected research approach was based on and limited to an organizational viewpoint and utilized organizational theories. Furthermore, the focus on this study was merely on organizations, and individual viewpoint was used only to complement an organizational perspective. This selection hence ignored literature presented in other relevant research areas, for example in psychology or in human resource development. These research areas could have enlightened the individual side of the SPI – i.e. personal improvement – more closely and could have brought more detailed knowledge on communication in software development. Considering the research focus, patterns as communication tools, the study was chosen to concentrate on particularly design patterns and not for example on architectural patterns or software components in general. This has further narrowed the scope of research. All of these premises on research theory, viewpoint and research focus were also reflected on data analysis.

Second, considering the conduct of the research, other limitations need to be acknowledged. The case companies of the research were all developing mobile phone applications. This was initially because of their own interest towards design patterns as development tools. Furthermore, their business environment

was evaluated as providing a narrow enough context for suggesting the practical benefits of design patterns. Also, the value chain and the specific characteristics of the business, as explained in the fourth and fifth articles, provided an interesting area for the research. This selection of these companies as case studies can also be seen as a limitation for the research, narrowing down the likely application of the research results. However, similar problem areas are still relevant in many other software organizations, as the results of the first article indicate. Additionally, this research setting provides an interesting topic for further research to be performed in a different business field.

Furthermore, selected research methods may have an effect on the research results. The results might have been significantly different, if, for example, only quantitative methods were used. Use of a qualitative approach and particularly the multiple case studies were selected to gain a rich view of the problem area, which could not have been reached with a different research approach. Also, the available resources have affected the research methods, particularly regarding data collection. Many interviews were carried out in groups, which might have affected the acquired information. For example, observation or involved action research might have supplied even richer data on the case companies. However, due to the limited time of both interviewers and particularly interviewees, this was not possible.

6.3 Further studies

Both the research process and results call for future studies. The difficulties that were encountered during this research often related specifically to the empirical evaluation of the research results. The companies can be slow to adopt process improvements and there is a delay before the actual benefits become visible. In addition, the companies generally consider specific practices as classified information, and thus they are not willing to share their plans or the results of improvements. Hence, an in-depth empirical evaluation of the research results remains for later research projects. Definitely, it would be interesting to do a follow-up on the case companies to measure the real business benefits that the initiated process improvements have provided.

In addition to empirical evaluations, also the case studies highlighted tool support for patterns. This topic was first raised in the study on tailoring patterns to organizational roles, where tool support was needed to format the presentation formats of patterns. Secondly, it seems that tool support is definitely needed in creation, maintaining and retrieving patters from the repository. And as it often is with software development, features and usability of the available tools (in addition to their price) can greatly affect the work practices themselves. Hence, this topic seems to be of utmost importance and further research is needed.

Furthermore, an interesting research area arose during this study regarding patterns as boundary objects in software development context. The benefit

of patterns is that they provide a vocabulary for difficult and complex topics. Concurrently, a well-written pattern provides different perspectives to a topic and uses a variety of presentation formats (e.g. diagrams, code, pictures and text). These pattern properties indicate that they could be even further exploited as boundary objects between different communities of practice in software organization. This evidently calls for further research on patterns' use in design collaboration, as is highlighted also in other studies (Lee, 2007).

YHTEENVETO (FINNISH SUMMARY)

Prosessien parantaminen on välttämätöntä kaikissa ohjelmistoyrityksissä. Parannukset edellyttävät yksilöiden oppimista ja opitun tiedon sisäistämistä osana organisaation toimintatapoja ja rakenteita, eli organisaation oppimista.

Tässä opinnäytetyössä ohjelmistoprosessien parantamista tutkitaan organisaation oppimisen näkökulmasta. Tarkoituksena on löytää käytännön keinoja prosessien parantamiseen, joten tutkimuskohteena on erityisesti tietämyksen hallinta ja tietämyksen jakaminen. Tutkimuksessa analysoidaan suunnittelumalleja eli patterneja ohjelmistojen suunnittelussa tarvittavan tiedon jakamisvälineenä.

Tutkimus sisältää viisi itsenäistä artikkelia ja kukin artikkeli lähestyy tutkimusaihetta eri näkökulmasta. Käytetyt tutkimusmenetelmät ovat sekä määrällisiä että laadullisia. Ensimmäisessä artikkelissa tutkittiin oppimista suomalaisissa ohjelmistotaloissa kyselymenetelmän ja tilastollisen analysoinnin avulla. Neljässä muussa artikkelissa käytettiin menetelminä tapaustutkimuksia ja kirjallisuustutkimusta.

Tutkimuksen tulokset osoittavat, että organisaation oppimista voidaan tukea tehokkaalla tiedon hallinnalla. Suunnittelumallit ovat yksi tapa dokumentoida, tallentaa ja jakaa ohjelmistojen suunnittelussa tarvittavaa osaamista. Lisäksi suunnittelumallit tarjoavat selkeät ja mitattavat tavoitteet prosessin parantamiseen organisaation eri tasoilla. Jotta suunnittelumalleja voitaisiin organisaatioissa hyödyntää mahdollisimman paljon, niiden käyttö on suunniteltava etukäteen ja sen tulee olla kiinteä osa päivittäistä ohjelmistojen kehitystyötä.

Tutkimuksen tuloksia voidaan hyödyntää ohjelmistoyrityksissä, kun mietitään keinoja ohjelmistokehittäjien oppimisen ja kommunikoinnin tukemiseen. Tiedeyhteisössä tutkimus täydentää patterneista jo tehtyä tutkimusta yhdistäen patternit uudella tavalla organisaation oppimiseen ja prosessien parantamiseen ja näin ollen tarjoaa uusia mielenkiintoisia ja hyödyllisiä tutkimuskohteita alueelta.

REFERENCES

Abrahamsson, P., Salo, O., Ronkainen, J. and Warsta, J. 2002. Agile software development and methods - review and analysis. VTT publications 478, Espoo.

Agerbo, E. and Cornils, A. 1998. How to preserve the benefits of Design Patterns. In 13th ACM SIGPLAN conference on Object-oriented programming, systems, languages and applications. Oregon, USA, ACM, 134-143.

Akgün, A.E., Lynn, G.S., and Reilly, R. 2002. Multi-dimensionality of learning in new product development teams, European Journal of Innovation Management 5(2), 57-72.

Alavi, M. and Leidner, D. E. 1999. Knowledge management systems: issues, challenges and benefits. Communications of the AIS, 1 (2).

Alavi, M. and Leidner, D. E. 2001. Review: Knowledge Management and Knowledge Management Systems: Conceptual Foundations and Research Issues. MIS Quarterly 25 (1), 107-136.

Alexander, C. 1979. The Timeless Way of Building, New York: Oxford University Press.

Alexander, C., Ishikawa, S., Silverstein, M., Jakobson, M., Fiksdahl-King, I., and Angel, S. 1977. A Pattern Language. (Volume 2). New York: Oxford University press.

Argote, L. 1999. Organizational learning: creating, retaining, and transferring knowledge. 1999. USA: Kluwer Academic Publishers.

Argyris, C. and Schön, D. 1978. Organizational Learning: A Theory of Action Perspective. Massachusetts: Addison-Wesley.

Assmann, D. and Punter, T. 2004. Towards partnership in software subcontracting, Computers in Industry 54, 137-150.

Astrachan, O., Berry, G., Cox, L. and Mitchener, G. 1998. Design Patterns: an essential component of CS Curricula. In 29th SIGCSE technical symposium on Computer science education 26.2. - 1.3.1998, Atlanta, Georgia, USA, ACM, 153-160.

Aurum, A., Daneshgar, F. and Ward, J. 2008. Investigating knowledge management practices in software development organizations - An Australian experience. Information and Software Technology 50, 511-533.

Babar, M. A., Gorton, I. and Jeffery, R. 2005. Capturing and Using Software Architecture Knowledge for Architecture-Based Software Development. In Proceedings of the 5th International Conference on Quality Software. IEEE, 169 - 176.

Basili, V. R. and Caldiera, G. 1995. Improve Software Quality by Reusing Knowledge and Experience. Sloan management review 37(1), 55–64.

Bhatt, G. D. 2001. Knowledge management in organizations: examining the interaction between technologies, techniques and people. Journal of Knowledge Management 5(1), 68-75.

Beck, K., Crocker, R., Meszaros, G., Coplien, J.O., Dominick, L., Paulisch, F., and Vlissides, J. 1996. Industrial experience with design patterns. In 18th International Conference on Software Engineering, Berlin, Germany, IEEE, 103-114.

Bennet, D. and Bennet, A. 2008. Engaging tacit knowledge in support of organizational learning. VINE: The journal of information and knowledge management systems 1(38) , 72-94.

Boehm, B. W. 1988. A Spiral Model of Software Development and Enhancement. Computer 21 (5), 61-72.

Booch, G. 2007. Object oriented analysis and design with applications. Third edition. Addison-Wesley: California.

Booch, G. 2008a. Architectural Organizational Patterns. IEEE Software 25(3), 18 - 19.

Booch, G. 2008b. Tribal Memory. IEEE Software 25(2), 16 - 17.

Boland, R. J. Jr. and Tenkasi R. V. 1995. Perspective Making and Perspective Taking in Communities of Knowing. Organization Science 6(4), 350-372.

Borchers, J. O. 2000. A Pattern approach to interaction design. In Conference on Designing interactive systems: processes, practices, methods, and techniques. USA: ACM Press, 369-378

Bowker, G. C. and Star, S. L. 1999. Sorting things out. Cambridge, MA: MIT Press.

Brietzke, J. and Rabelo, A. 2006. Resistance factors in software process improvement. Clei Electronic Journal 9 (1).

Buschmann, F., Henney, K. and Schmidt, D.C. 2007. Past, Present, and Future Trends in Software Patterns, IEEE Software 24(4), 31 - 37.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M. 1996. Pattern-oriented software architecture - A system of patterns. Chicester, West Sussex, England: John Wiley & Sons ltd.

Calantone, R. J., Cavusgil, S. T., and Zhao, Y. 2002. Learning orientation, firm innovation capability, and firm performance. Industrial Marketing Management, 31, 515–524.

Carlile, P. R. 2002. A Pragmatic View of Knowledge and Boundaries, Organization Science 13(4), 442-455.

Carnegie Mellon University, 2009. CMMI Process Maturity Profile March 2009 Report. [Online] Retrieved 10th January 2010 from: http://www.sei.cmu.edu/appraisal-program/profile/pdf/CMMI/2009MarCMMI.pdf

Chiu, M.-L. 2002. An Organizational View of Design Communication in Design Collaboration. Design Studies (23), 187-210.

Cline, M.P. 1996. The pros and cons of adopting and applying design patterns in the real world. Communications of the ACM 39 (10), 47-49.

CMMI for Development. 2006.Version 1.2. Technical Report. CMU/SEI-2006-TR-008. Software Engineering Institute (SEI), Pittsburgh. [Online] Retrieved 27th September 2009 from: http://www.sei.cmu.edu/pub/documents/06.reports/pdf/06tr008.pdf

Cockburn, A. 2001. Agile software development, the people factor. Computer 34(11), 131-133.
Collin, K. 2005. Experience and Shared Practice - Design Engineers' Learning at work. University of Jyväskylä, Doctoral Dissertation.
Conradi, R. and Fuggetta, A. 2002. Improving Software Process Improvement. IEEE Software 19(4), 92-100.
Coplien, J. O. 2000. Software Patterns, New York: SIGS Books & Multimedia.
Coplien, J. O. and Schmidt, D. C. (Eds.). 1995. Pattern languages of program design. New York: ACM Press/Addison-Wesley.
Crossan, M. M., Lane, H. W. and White, R. E. 1999. An organizational learning framework: from intuition to institution. Academy of management review 24 (3), 522-537.
Curtis, B., Krasner, H. and Iscoe, N. 1988. A field study of the software design process for large systems. Communications of the ACM 31 (11), 1268-1287.
Davenport, T. H. and Prusak, L. 1998. Working knowledge: How Organizations Manage What They Know. Boston, Massachusetts, USA: Harvard Business School Press.
Deming, W. E. 1994. Out of the Crisis: quality, productivity and competitive position. England, Cambridge University Press.
DeSouza, K. C. 2003. Barriers to Effective Use of Knowledge Management Systems in Software Engineering. Communications of the ACM 46 (1), 99-101.
Diaz, M. and Sligo, J. 1997. How software process improvement helped Motorola. IEEE Software 14 (5), 75-81.
Dixon, D. 2009. Pattern Languages for CMC Design. In B. Whitworth, and B. de Moor, (Eds.) Handbook of Research on Socio-Technical Design and Social Networking Systems. Information Science Reference, USA.
Dorling, A. 1993. SPICE: Software Process Improvement and Capability dEtermination. Software Quality Journal 2, 209-224.
Dybå, T. 2001. Enabling Software Process Improvement: An Investigation of the Importance of Organizational Issues. Department of Computer and Information Science, Faculty of Physics, Informatics and Mathematics. Norwegian University of Science and Technology, Doctoral dissertation,
Dybå, T. 2005. An Empirical Investigation of the Key Factors for Success in Software Process Improvement. IEEE Transactions on Software Engineering 31 (5), 410 - 424.
Dybå, T and Dingsoyr, T. 2008. Empirical studies of agile software development: A systematic review. Information and Software Technology 50(9-10), 833-859.
Easterby-Smith, M. 1997. Disciplines of organizational learning: contributions and critiques. Human Relations 50 (9), 1085-1113.
Easterby-Smith, M. and Lyles, M. A. 2005. The Blackwell Handbook of Organizational Learning and Knowledge Management. Blackwell Publishing, USA.

Easterby-Smith, M., Crossan, M. and Nicolini, D. 2000. Organizational learning: debates past, present and future. Journal of management studies 37 (6), 783–796.

Eckert, C and Boujout, J-F. 2003.The Role of Objects in Design Co-Operation: Communication through Physical or Virtual Objects. Computer Supported Cooperative Work 12, 145–151.

El Emam, K., Drouin, J-N. and Melo, W. 1998. Spice: The Theory and Practice of Software Process Improvement and Capability Determination. IEEE Computer Society Press: California.

Erickson, T. 2000. Lingua Francas for Design: Sacred Places and Pattern Languages. In the proceedings of Designing interactive systems: processes, practices, methods and techniques. New York: ACM Press.

Fach, P.W. 2001. Design reuse through frameworks and patterns. IEEE Software 18(5), 9-10.

Fowler, M. 2003. Patterns of Enterprise Application Architecture, Boston: Addison-Wesley.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. 1995. Design Patterns: Elements of reusable object oriented software, Boston, USA: Addison-Wesley.

Gasston, J. and Halloran, P. 1999. Continuous Software Process Improvement Requires Organizational Learning: An Australian Case Study. Software Quality Journal (8), 37-51.

Goldfedder, B. and Rising, L. 1996. A Training Experience with Patterns. Communications of the ACM 39 (10), 60-64.

Gschwind, T., Koehler, J. and Wong, J. 2008. Applying Patterns during Business Process Modeling. In M. Dumas, M. Reichert and M.-C. Shan (Eds.): BPM 2008, LNCS 5240, Berlin Heidelberg, Springer-Verlag, 4–19.

Hagen, M. 2002. Support for the definition and usage of process patterns. In Seventh European Conference on Pattern Languages of Programs. Position paper. [Online] Retrieved 27th August 2010 from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.5.5462&rep=rep1&type=pdf

Haley, T.J. 1996. Software process improvement at Raytheon. IEEE Software 13 (6), 33 – 41.

Hansen, M. T., Nohria, N. and Tierney, T. 1999. What's Your Strategy for Managing Knowledge? Harvard Business Review 77 (2), 16-16.

Harjumaa, L. 2005. Improving the software inspection process with patterns. University of Oulu, Academic dissertation.

Harter, D. E., Krishnan, M. S. and Slaughter, S. A. 2000. Effects of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development. Management Science 46 (4), 451-466.

Heikkilä, M. 2009. Learning and Organizational Change in SPI Initiatives. In F. Bomarius et al. (Eds.): PROFES 2009, LNBIP 32, Berlin Heidelberg, Springer-Verlag, 216–230.

Hendry, D. G. 2004. Communication functions and the adaptation of design representations in interdisciplinary teams. In D. Benyon & P. Moody (Eds.) Proceedings of DIS2004. New York: ACM Press, 123-132.

Herbsleb, J., Mockus, A., Finholt, T. A and Grinter, R. E. 2001. An empirical study of global software development: distance and speed. In Proceedings of the 23rd International Conference on Software Engineering. USA: IEEE Computer Society, 81 - 90.

Herbsleb, J., Zubrow, D., Goldenson, D., Hayes, W. and Paulk, M. 1997. Software quality and the Capability Maturity Model. Communications of the ACM 40 (6), 30-40.

Herrmann, T., Hoffmann, M., Jahnke, I., Kienle, A., Kunau, G., Loser, K. and Menold, N. 2003. Knowledge management II: Concepts for usable patterns of groupware applications. In International ACM SIGGROUP Conference on Supporting Group Work, Sanibel Island, Florida, USA, ACM Press.

Herzum, M. 2002. The importance of trust in software engineers' assessment and choice of information sources. Information and organization 12, 1-18.

Hoegland, M. and Gemuenden, G. 2001. Teamwork Quality and the Success of Innovative Projects. Organization Science 12 (4), 435-449.

Holden, R., Smith, V. and Devins, D. 2003. Using 'emloyee-led development' to promote lifelong learning in SMEs: a research note. Human Resource Development International 6 (1), 125-132.

Hollenbach, C., Young, R., Pflugrad, A. and Smith, D. 1997. Combining quality and software improvement. Communications of the ACM 40 (6), 41-45.

Huber, G.P. 1991. Organizational learning: The contributing processes and the literatures. Organization science 2 (1), 71-87.

Humphrey, W.S. 1989. Managing the software process. SEI series in software engineering. Massachusetts, USA: Addison-Wesley.

Humphrey, W.S. 1995. A discipline for software engineering. Reading (MA): Addison-Wesley.

Humphrey, W.S., Kitson, D. and Kasse, T. 1989. The state of software engineering practice: A preliminary report. In Proceedings of the 11th International Conference on Software Engineering (ICSE). USA: ACM, 277-288.

Humphrey, W.S., Over, J. W., Konrad, M. C. and Peterson, W. C. 2007. Future Directions in Process Improvement. Crosstalk - The journal of defense software engineering, 20 (2), 17-22.

Huysman, M. 2000. Rethinking the organizational learning: analyzing learning processes of information system designers. Accounting, Management and Information Technology 10, 81-99.

Iivari, J. and Huisman, M. 2007. The Relationship Between Organizational Culture and the Deployment of Systems Development Methodologies. MIS Quarterly 31(1), 35-58.

Irani, Z., Sharif, A. M. and Love P.E.D. 2009. Mapping knowledge management and organizational learning in support of organizational memory. International Journal of Production Economics 122, 200–215.

Jacobson, I., Griss, M. and Jonsson, P. 1997. Software Reuse: Architecture, Process and Organization for Business Success, New York: ACM Press.

Jennex, M. E. and Olfman, L. 2002. Organizational memory/ knowledge effects on productivity - A longitudinal study. In Proceedings of the 35th Hawaii International Conference on System Sciences, IEEE, 1029 - 1038.

Jennex, M. E. and Olfman, L. 2004. Assessing Knowledge Management Success/Effectiveness Models. In Proceedings of the 37th Hawaii International Conference on System Sciences, IEEE Computer Society.

Johansen, J. and Pries-Heje, J. 2007. Success with Improvement - Requires the Right Roles to be Enacted - in Symbiosis. Software Process Improvement and Practice 12, 529-539.

Järvinen, P. and Järvinen, A. 2000. Tutkimustyön metodeista. Tampere: Opinpajan kirja.

Kakabadse, N. K., Kakabadse, A. and Kouzmin, A. 2003. Reviewing the Knowledge Management Literature: Towards a Taxonomy. Journal of Knowledge Management 7 (4), 75-91.

Kankanhalli, A., Tan, B.C.Y. and Wei, K-K. 2005. Contributing Knowledge To Electronic Knowledge Repositories: An Empirical Investigation, MIS Quarterly 29 (1), 113-143.

Ketola, P. 2002. Integrating Usability with Concurrent Engineering in Mobile Phone Development. Department of Computer and Information Sciences: University of Tampere, Tampere. Academic Dissertation.

Kettunen, P. 2003. Managing embedded software project team knowledge. IEEE Proceedings Software 150 (6), 359–366.

Kim, D.H. 1993. The link between individual and organizational learning. Sloan management review 35(1), 37–50.

Koc, T. 2007. Organizational determinants of innovation capacity in software companies. Computers & Industrial Engineering 53, 373–385.

Kock, N. 1999. Process Improvement and Organizational Learning: The Role of Collaboration Technology, London: Idea group Publishing.

Kolb, D. A. 1984. Experiential learning. Experience as the source of learning and development. New Jersey: Prentice-Hall.

Komi-Sirviö, S., Mäntyniemi, A. and Seppänen, V. 2002. Toward A Practical Solution for Capturing knowledge for Software Projects. IEEE Software 19(3), 60-62.

Korkala, M. and Abrahamsson, P. 2007. Communication in Distributed Agile Development: A Case Study. In Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications, USA: IEEE Computer Society, 203 - 210.

Korpela, M., Mursu, A. and Soriyan, H.A. 2002. Information Systems Development as an Activity. Computer Supported Cooperative Work 11, 111-128.

Kotlarsky, J. and Oshri, I. 2005. Social ties, knowledge sharing and successful collaboration in globally distributed system development porjects. European Journal on Information Systems 14, 37-48.

Kramer, J. 2007. Is abstraction the key to computing? Communications of ACM 50 (4), 37-42.

Kramer, J. and Hazzan, O. 2006. The role of abstraction in software engineering. In Proceedings of the 28th international conference on Software engineering. USA: ACM, 1017 – 1018.

Krasner, H. 2001. Accumulating the Body of Evidence for The Payoff of Software Process Improvement. Software Process Improvement, IEEE Computer Society Press, 519-539.

Kucza, T., Nättinen, M., and Parviainen, P. 2001. Improving knowledge management in software reuse process. In F. Bomarius, S. Komi-Sirviö (Eds.) Product Focused Software Process Improvement. Berlin: Springer, 141-152.

Kulpa, M. K. and Johnson, K. A. 2008. Interpreting the CMMI: A Process Improvement Approach. Second edition. USA: Auerbach Publications.

Lee, C. P. 2007. Boundary Negotiating Artifacts: Unbinding the Routine of Boundary Objects and Embracing Chaos in Collaborative Work. Computer Supported Cooperative Work 16, 307-339.

Lee, H. and Choi, B. 2003. Knowledge Management Enablers, Processes, and Organizational Performance: An Integrative View and Empirical Examination. Journal of Management Information Systems 20(1), 179-228.

Leung, H. K. N. 1999. Slow change of information system development practice. Software Quality Journal 8(3), 197-210.

Lindvall, M. and Rus, I. 2000. Process Diversity in Software Development. IEEE Software 17(4), 14-18.

Lindvall, M., Rus, I. and Sinha, S.S. 2003. Software systems support for knowledge management. Journal of knowledge management 7(5), 137-150.

Lyytinen, K. and Robey, D. 1999. Learning failure in information systems development. Information Systems Journal (9), 85-101.

Malone, E., Leacock, M. and Wheeler, C. 2005. Implementing a pattern library in the real world. A Yahoo! case study. [Online]. Retrieved 15th February 2010 from http://leacock.com/patterns/leacock_malone_wheeler.pdf

Marquardt, K. 2002. What Makes Pattern Languages Work Well. Position paper in Seventh European Conference on Pattern Languages of Programs. [Online]. Retrieved 17th June 2010 from http://www.haase-consulting.com/workshops/FgEuroplop02/WhatMakesPatternLanguagesWork_KlausMarquardt.pdf

Mathiassen, L., Nielsen, P.A., Pries-Heje, J. 2002. Learning SPI in practice. In L. Mathiassen, J. Pries-Heje, and O. Ngwenyuma (Eds): Improving Software Organizations: From Principles to Practice, USA: Addison-Wesley, 3-21.

Mathiassen, L. and Pedersen, K. 2005. The dynamics of knowledge in systems development practice. In Proceedings of the 38th Annual Hawaii International Conference on System Sciences. IEEE, 233a - 233a.

Mathiassen, L. and Pourkomeylian, P. 2003. Managing knowledge in a software organization. Journal of Knowledge Management 7 (2), 63-80.

May, D. and Taylor, P. 2003. Knowledge management with patterns. Communications of the ACM 43 (7), 94-99.

McFeeley, B. 1996. IDEAL: A User's Guide for Software Process Improvement. CM/SEI-96-HB-001, Software Engineering Institute, USA.

Monge, P.R., Bachman, S.G., Dillard, J.R. and Eisenberg, E.M. 1982. Communication competence in the workplace: model testing and scale development. In M. Burgoon (Ed.) Communication yearbook, vol. 5. USA: Transaction Books, 505-528.

Montoni, M. A., Cerdeiral, C., Zanetti, D. and da Rocha, A. R. C. 2008. A Knowledge Management Approach to Support Software Process Improvement Implementation Initiatives. In R. V. Connor, N. Baddoo, K. Smolander and R. Messnarz (Eds.) Proceedings of 15th European Conference on Software Process Improvement. Berlin: Springer, 164-175.

Nan, N. and Harter, D. E. 2009. Impact of Budget and Schedule Pressure on Software Development Cycle Time and Effort. IEEE Transactions on Software Engineering 35 (5), 624-637.

Nasir, M.H.N.M., Ahmad, R. and Hassan, N.H. 2008. An empirical study of barriers in the implementation of software process improvement project in Malaysia. Journal of Applied Sciences 8 (23), 4362-4368.

Ngwenyama, O. and Nielsen, P. A. 2003. Competing Values in Software Process Improvement: An Assumption Analysis of CMM From an Organizational Culture Perspective. IEEE Transactions on Engineering Management 50 (1), 100-112.

Niazi, M. 2006. Software Process Improvement: A Road to Success. In J. Münch, and M. Vierimaa (Eds.): PROFES 2006, LNCS 4034. Berlin: Springer-Verlag, 395 – 401

Niazi, M. 2009. Software process improvement implementation: avoiding critical barriers. Crosstalk - the journal of defense software engineering, 22(1), 24-27.

Niazi, M., Wilson, D. and Zowghi, D. 2005. A maturity model for the implementation of software process improvement: an empirical study. The Journal of Systems and Software 74 (2), 155-172.

Niazi M., Wilson, D., Zowghi, D. 2006. Critical success factors for software process improvement implementation: an empirical study. Software Process: Improvement and Practice. Special Issue on Free or Open Source Software Development (F/OSSD) Projects 11(2), 193 - 211.

Nonaka, I. 1994. A Dynamic Theory of Organizational Knowledge Creation. Organization science 5 (1), 14-37.

Nonaka, I. and Takeuchi, H. 1995. The Knowledge creating company. Oxford: Oxford University Press.

Nonaka, I. and Toyama, R. 2003. The Knowledge-creating theory revisited: knowledge creation as a syntehisizing process. Knowldge Management Research & Practice 1, 2-10.

O'Hara, F. 2000. European Experiences with Software Process Improvement. In Proceedings of the 22nd international conference on Software engineering. USA: ACM, 635 – 640.

Oktaba, H. and Piattini, M. (Eds.) 2008. Software process improvement for small and medium enterprises - Techniques and Case studies. USA: Information Science Reference.

Olson, D. 1998. FAQ from Bruce Anderson's group at ChiliPLoP'98. Wickenburg AZ, 17-20 March 1998. [Online] Retrieved 23th April 2006 from http://c2.com/cgi/wiki?SomePatternsQuestionsAnswered

Paasivaara, M. 2003. Communication Needs, Practices and Supporting structures in Global Inter-Organizational Software Development Projects. In ICSE '03 International Workshop on global Software Development, IEEE Computer Society.

Perez Lopez, S., Montes Peon, J. M. and Vasquez Ordas, C. J. 2005. Organizational learning as a determining factor in business performance. The Learning Organization 12 (3), 227-245.

Popper, M. and Lipshitz, R. 2000. Organizational Learning: Mechanisms, Culture, and Feasibility. Management Learning 31, 181-196.

Porter, R. and Clader, P. 2004. Patterns in Learning to Program - An Experiment? In Sixth conference on Australian computing education. Australia: Australian Computer Society, 241 – 246.

Pressman, R.S. 2000. Software Engineering - A Practitioner's Approach. European Adaptation. Berkshire, England: McGraw-Hill Publishing Company.

Pries-Heje, J. and Johansen, J. (Eds.) 2010: Software Process Improvement Manifesto. Version A.1.2.2010.

Proulx, V.K. 2000. Programming Patterns and Design Patterns in the introductory computer science course. In 31st SIGCSE Technical Symposium on Computer Science Education. USA: ACM, 80-84.

Prusak, L. 2001. Where did knowledge management come from? IBM Systems Journal, 40(4), 1002 - 1007.

Rainer, A. and Hall, T. 2002. Key success factors for implementing software process improvement: a maturity-based analysis. The Journal of Systems and Software 62, 71–84.

Ravichandran, T. and Rai, A. 2003. Structural analysis of the impact of knowledge creation and knowledge embedding on software process capability. IEEE Transactions on Engineering Management 50(3), 270-284.

Rico, D. F. 2004. ROI of Software Process Improvement : For Project Portfolio Managers and PMO's. USA: J. Ross Publishing Inc.

Robey, D., Wishart, N.A. and Rodriquez-Diaz, A.G. 1995. Merging the metaphors for organizational improvement: Business process reengineering as a component of organizational learning. Accounting, Management & Information Technology 5 (1), 23-39.

Robillard, P. N. 1999. The role of knowledge in software development. Communications of the ACM 42(1), 87-92.

Rus, I. and Lindvall, M. 2002. Knowledge Management in Software Engineering. IEEE Software 19 (3), 26-38.

Saiedian, H. and Chennupati, K. 1999. Towards an evaluative framework for software process improvement models. The Journal of Systems and Software 47, 139-148.

Salo, O. and Abrahamsson, P. 2008. Agile methods in European embedded software development organisations: a survey on the actual use and usefulness of Extreme Programming and Scrum. IEE Software 2(1), 58–64.

Sametinger, J. 1997. Software engineering with reusable components, Berlin: Springer-Verlag.

Sarkkinen, J. 2006. Design as discourse. Representation, representational practice and social practice. Jyväskylä Studies in Computing 62. Department of Information Science, University of Jyväskylä. Doctoral dissertation

Schmidt, D.C. 1995. Using Design Patterns to Develop Reusable Object Oriented Communication Software. Communications of the ACM, 38(10), 65-74.

Schmidt, D.C. and Cleeland, C. 2000. Applying a Pattern Language to develop Extensible ORB Middleware. In L. Rising (Ed.): Design patterns in communications software. England: Cambridge University Press, 393-438.

Schmidt, D.C., Fayad, M., Johnson, R.E. and Guest Editors 1996. Software Patterns. Communications of the ACM 39 (10), 37-39.

Schmidt, D.C., Stal, M., Rohnert, H., and Buschmann, F. 2000. Pattern-oriented software architecture - Patterns for concurrent and networked objects, Chichester, England: John Wiley and Sons.

Schummer, T. and Lukosch, S. 2007. Patterns for Computer Mediated Interaction. Wiley Software Patterns Series. West Sussex, England: Wiley.

Schönström, M. 2005. A knowledge process perspective on the Improvement of Software Process. Department of Informatics, Lund University. Academic dissertation.

Segal, J. 2001. Organizational Learning and Software Process Improvement: A case study.In 3rd International Workshop on Learning Software Organizations. Germany, Springer-Verlag, 68-82.

Seigerroth, U. and Lind, M. 2006. Facilitating Learning in SPI through Co-design. In G.A. Nilsson, R. Gustas, W. Wojtkowski, W.G. Wojtkowski, S. Wrycza, and J. Zupancic (Eds.): Advances in Information Systems Development. Bridging the gap between academia and industry. USA: Springer, 119-130.

Seleim, A., Ashour, A. and Bontis, N. 2007. Human capital and organizational performance: a study of Egyptian software companies. Management Decision 45(4), 789-801.

Senge, P.M. 1990. The fifth discipline: the art and practice of the learning organization, New York: Doubleday Currency.

Sharratt, M. and Usoro, A. 2003. Understanding Knowledge-Sharing in Online Communities of Practice. Electronic Journal on Knowledge Management 1 (2), 187-196.

Slaughter, S. A. and Kirsch, L. J. 2006. The Effectiveness of Knowledge Transfer Portfolios in Software Process Improvement: A Field Study. Information Systems Research 17(3), 310-320.

Smite, D. 2005. A Case Study: Coordination Practices in Global Software Development, Product Focused Software Process Improvement. In 6th International Conference, Berlin, Springer

Sodhi, J. and Sodhi, P. 1998. Software Reuse: Domain Analysis and Design Process, New York: McGraw-Hill.

Sommerville, I. 2001. Software engineering. 6th edition. Harlow: Addison-Wesley.

Soundarajan, N., Hallstrom, J.O., Shu, G. and Delibas, A. 2008. Patterns: from system design to software testing. Innovations System Software Engineering 4, 71-85.

Stake, R. E. 2000. Case studies. In N. K. Denzin and Y. S. Lincoln (Eds.) Handbook of Qualitative Research. California: Sage Publications, 443-466.

Standish Group. 2009. Chaos Report.

Staples, M., Niazi, M., Jeffery, R., Abrahams, A., Byatt, P. and Murphy, R. 2007. An exploratory study of why organizations do not adopt CMMI. The Journal of Systems and Software 80, 883–895.

Star, S.L. and Griesemer, J.R. 1989. Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39. Social Studies of Science 19 (3), 387-420.

Star, S. L. and Ruhleder, K. 1996 In J. Yates and van J. Maanen, (Eds.) 2001. Information technology and organizational transformation: History, rhetoric and practice. Usa: Sage Publications.

Stein, E.W. 1995. Organizational Memory: Review of Concepts and Recommendations for Management. International Journal of Information Management 15 (1), 17-32.

Stelzer, D. and Mellis, W. 1998. Success Factors of Organizational Change in Software Process Improvement. Software Process Improvement and Practice 4, 227–250.

Subrahmanian, E., Monarch, I., Konda, S., Granger, H., Milliken, R., Westerberg, A. and the N-DIM Group. 2003. Boundary objects and prototypes at the interfaces of engineering design. Computer Supported Cooperative Work 12, 185-203.

Suscheck, C. A. and Ford, R. 2008. Jazz improvisation as a learning metaphor for the scrum software development methodology. Software Process Improvement and Practice 13 (5), 439–450.

Tenopir, C. and King, D.W. 2004. Communication Patterns of Engineers. Hoboken, NJ, USA: John Wiley & Sons.

Terveen, L.G., Selfridge, P.G., and Long, M.D. 1993. From "Folklore" to "Living Design Memory". In SIGCHI conference on Human factors in computing systems. Amsterdam: ACM Press, 15 - 22.

Trienekens, J. J. M., Kusters, R. J., van Genuchten, M. J. I. M. and Aerts, H. 2008. Targets, drivers and metrics in software process improvement: Results of a

survey in a multinational organization. Software Quality Journal 16, 237–261.
Turner, J. R., Keegan, A. and Crawford, L. 2000. Learning by experience in the project-based organization. Report series Research in Management. ERS-2000-58-ORG. University of Rotterdam.
van Solingen, R. 2004. Measuring the ROI of Software Process Improvement. IEEE Software 21 (3), 32 - 38.
van Solingen, R. 2009. A Follow-Up Reflection on Software Process Improvement ROI. IEEE Software 26(5), 77-79.
van Solingen, R., Berghout, E., and Trienekens, J. 2000. From process improvement to people improvement: enabling learning in software development. Information and Software Technology 42, 965-971.
Vera, D. and Crossan, M. 2003. Organizational learning and knowledge management: Toward an integrative framework. In M. Easterby-Smith, and M. Lyles, (Eds.) The Blackwell Handbook of Knowledge Management and Organizational Kearning. Oxford, UK: Blackwell Publishing, 122-141.
Vesiluoma, S. 2009. Understanding and supporting knowledge sharing in software engineering. Tampere University of Technology, Publication 843. Doctoral disseration,
Walsh, J.P., and Ungson, G.R. 1991. Organizational memory. Academy of management review 16(1), 57-91.
Walz, D.B., Elam, J.J., and Curtis, B. 1993. Inside Software Design Team: Knowledge Acquisition, Sharing and Integration. Communications of the ACM 36 (10), 63-77.
Waterson, P. E., Clegg, C. W., Axtell, C.M. 1997. The dynamics of work organization, knowledge and technology during software development. International Journal of Human-Computer Studies 46 (1), 79-101.
Wenger, E. 2003. Communities of Practice and Social Learning systems. In D. Nicolini, S. Gherardi and D., Yanow (Eds.) Knowing in Organizations. New York: M.E.Sharpe, 225-246.
Wilkie, F., McFall, D., McCaffery, F., 2005. An evaluation of CMMI process areas for small- to medium-sized software development organisations. Software Process Improvement and Practice 10 (2), 189–201.
Wing, J. M. 2006. Computational thinking. Communications of the ACM 49(3), 33-35.
Wirfs-Brock, R. J. 2009. Designing with an agile attitude. IEEE Software 26 (2), 68-69.
Wong, K. Y. 2005. Critical success factors for implementing knowledge management in small and medium enterprises. Industrial Management & Data Systems 105(3), 261-279.
Wright, D. R. 2007. The Decision Pattern: Capturing and Communicating Design Intent. In Proceedings of the 25th Annual ACM International Conference on Design of Communication, USA
Yacoub, S.M. and Ammar, H.H. 2004. Pattern-Oriented Analysis and Design. Composing Patterns to Design Software Systems. Boston: Addison-Wesley.

Yang, J. 2007. The impact of knowledge sharing on organizational learning and effectiveness. Journal of Knowledge Management 11 (2), 83-90.

Yin, R. K. Case Study Research - Design and methods.3rd edition. Newbury Park: Sage Publications, 2003.

Zahran, S. 1998. Software Process Improvement. Practical guidelines for business success. Harlow: Addison-Wesley.

Zdrahal, Z., Mulholland, P., Domingue, J. and Hatala, M. 2000. Sharing engineering design knowledge in a distributed environment. Behaviour & Information Technology 19(3), 189-200.

Zhong, J. and Majchrzak, A. 2004. An Exploration of Impact of Cognitive Elaboration on Learning in ISD Projects. Information Technology and Management 5, 143-159.

ORIGINAL PAPERS

I

HOW ORGANIZATIONS SUPPORT LEARNING IN PRACTICE: A SURVEY ON FINNISH SOFTWARE ORGANIZATIONS

by

Riikka Ahlgren & Samuli Pekkola, 20.8 2010

Submitted to Journal of Information and Knowledge Management.

II

USING GROUPWARE TO FACILITATE ORGANIZATIONAL LEARNING AND SOFTWARE PROCESS IMPROVEMENT – A CASE STUDY

by

Riikka Ahlgren, Mirja Pulkkinen, Eleni Berki & Marko Forsell, 11.10.2006

Proceedings of European Systems & Software Process Improvement and Innovation (EuroSPI), pages 2.1-2.7

III

FACILITATING DESIGN KNOWLEDGE MANAGEMENT BY TAILORING SOFTWARE PATTERNS TO ORGANIZATIONAL ROLES

by

Riikka Ahlgren, 9.8.2007

Proceedings of 13th Americas Conference on Information Systems (AMCIS 2007), paper 463.

FACILITATING DESIGN KNOWLEDGE MANAGEMENT BY TAILORING SOFTWARE PATTERNS TO ORGANIZATIONAL ROLES

Riikka Ahlgren
Information Technology Research Institute, University of Jyväskylä
P.O. Box 35, FIN-40014 University of Jyväskylä, Finland
ahlgren@titu.jyu.fi

Abstract

This paper considers knowledge management in software development from organizational roles' viewpoint. The focus is on software patterns, which are seen as means to facilitate the design knowledge management. In this paper, patterns are promoted by tailoring their presentation according to the specific needs of each role. Roles' needs are justified by a case study made in a Finnish software organization. Pattern presentation formats are analyzed considering different skills of roles. The analysis reveals that different roles can best utilize different parts of pattern descriptions. As a result, a tentative model is built where relationships between roles and pattern parts are depicted. The result can be used to promote patterns as means for managing the design knowledge in organizations.

Keywords: *Software patterns, Knowledge management, Organizational Roles*

Introduction

Software organizations possess a vast amount of knowledge about their business and the technologies they use. Design knowledge, together with business knowledge and technical skills, form the core of a software organization's knowledge to have and to exploit (Stein 1995). Several knowledge management activities have been identified, which support the software development process (Rus et al. 2002). Suggested activities include for example document management, competence management, and software reuse (Rus et al. 2002). This paper utilizes and builds on the reuse perspective by considering patterns and their reuse as a knowledge management problem, as suggested by Highsmith (2003).

The contents of design knowledge are composed of the design trade-offs between the various software qualities, which are due to particular structural or behavioral aspects of the system (Gross et al. 2001). The difference between the terms design knowledge management and design reuse can be clarified by defining the knowledge management as reusing knowledge including both explicit and tacit knowledge, while the concept of reuse is merely focused on reusing explicit components (Kucza et al. 2001). Approaching the knowledge management with Nonaka's terms (Nonaka 1994), the focus of this research lays on the knowledge combina-

tion phase of the knowledge creation cycle, where design knowledge is being shared between the roles participating the development process.

Recent literature identifies several problems related to sharing of knowledge in software development (DeSouza 2003; Rus et al. 2002; Selfridge et al. 1992). The problems can be categorized as technological, organizational and individual problems (Rus et al. 2002). A general organizational problem is the lack of common language of developers and other roles (Buschmann et al. 1996; Walz et al. 1993), which is due to their different backgrounds, skills and interests. This problem becomes visible for example, when estimating the amounts of work for each project task. The developers should be able to present their work estimations in an understandable way for the project manager. Since the project manager does not necessarily have deep enough technical knowledge, the common vocabulary may be difficult to find.

However, the management of design knowledge is more often focused on tools (DeSouza 2003; Gomes et al. 2006; Kucza et al. 2001) than on the people and process perspective (Kucza et al. 2001). Hence, to get a broader view on the people perspective on knowledge management, we focus on organizational roles and particularly on project managers. This focus is motivated by the organizational setting, where software is built, namely project business. Project managers are the key in deciding how much time the developers can use to each of the project tasks. Thus, to enable better management of design knowledge, and further to be able to allocate time for the pattern related activities, the project managers must be aware of the design issues at least in general level.

Software patterns have been suggested to facilitate management of design knowledge and software reuse, since they capture both the design decisions made but also the decision rationale in a compact, easily-available and structured format (Buschmann et al. 1996). Further, previous research provides empirical evidence on patterns' benefits regarding design quality and communication improvement inside the development team (Beck et al. 1996; Cline 1996) and hence, this paper focuses on patterns as means for design knowledge management.

The research question addressed in this paper is "*How to make the patterns more understandable to the business-oriented roles of software development?*" The paper analyses pattern descriptions according to the skills needed for understanding and learning of each part of the description. The aim is to enable different roles better to exploit the patterns, and thus to facilitate the management of design knowledge during the software development. The tasks and the skills that are needed for various roles are considered, the presentation formats of patterns are analyzed, and the skills needed to understand different pattern parts are identified.

The paper is organized as follows. The following section describes the case setting and the research method. Software patterns as knowledge entities are presented after that, which is then followed by the organizational roles and their tasks. Pattern presentation formats are demonstrated and the pattern parts analyzed. Finally, tentative framework combining roles and different parts of patterns are presented and conclusions to the paper are provided.

Case Description and Research Method

The studied case organization is a large Finnish software company specialising in integrated ICT solutions and mobile products and services. The company stresses quality and efficient working habits, thus processes, quality assurance and time management are effectively implemented. Software processes as well as related practices are comprehensively documented.

Design knowledge management has been one of the company's improvement targets and during the past years several improvement efforts have been implemented. However, in daily development work the use of software patterns has merely been up to individual developers more than a common practice. As a result, the management of design knowledge has not considerably improved. Studying the patterns on free-time, as suggested by previous research (Cline 1996), neither improved the management of design knowledge.

In the case company, the unrealization of some improvements was mainly due the insufficient organizational support. For example, the time management system did not recognize the design knowledge management activities as tasks to be tracked. Furthermore, templates for project planning did not support those activities in adequate manner, thus project managers did not allocate time to the work efforts needed.

To make the management of design knowledge more efficient and common, and further to enable reuse of software artifacts, the company desired to raise the abstraction level of reuse and became interested in software patterns. The abstractness of the design information was supposed to make it more understandable for more roles and thus to increase the organizational support for design knowledge management.

However, presenting patterns did not solve the original problems: the patterns use was not supported by project management and by time allocation. Project managers and responsibles of time management system did not recognize patterns or their benefits. Thus, there was a need to convince project managers and others in non-technical roles on the advantages of patterns. This motivates the research reported in this paper.

The case study was made during winter 2005/2006. Our research method comprised semi-structured interviews, meeting discussions and an extensive document analysis. The analyzed documents included a quality handbook, process descriptions and templates for different purposes. Semi-structured interview method was chosen to keep the interview informal and conversational, and thus to encourage the interviewees to talk about also the difficulties encountered. In all sessions the researcher led the discussion into the topic with a short introduction.

Designers, architects and managers were interviewed in three separate sessions. First, the quality manager, who also worked as a project manager, was interviewed in order to identify the company's objectives of the pattern promotion. Then an extensive documentation analysis was made to identify the roles to be interviewed and the skills needed by different roles in their tasks. The aim of the second and third interviews was to verify the needs of the roles and the required skills, which had been identified from the documentation. The interviews lasted 30 and 45 minutes with eight programmers, designers or architects present in both sessions. There were 12 interviewees in total, for some persons attended both sessions.

In addition to the interviews, notes were taken in three other meetings, where patterns were presented and discussed. In the first meeting there were 6 designers and the technology manager. The last two meetings were only for company management. In addition to the technology manager of the case company, from four other companies there attended managers interested in promoting patterns.

Patterns as Knowledge Entities

Software patterns are reusable knowledge entities, more concrete than components but less concrete than frameworks (Fayad et al. 1999). A classical definition for a pattern is "a three part rule, which expresses the relation between a certain context, problem and solution" (Alexander 1979). They can be classified to idioms, design patterns and architectural patterns (Gamma et al. 1995) and are recognized tool to learn, document and to share experimental design knowledge (Buschmann et al. 1996; Gamma et al. 1995). In software development the patterns are experienced as precise enough to preserve the domain-specific knowledge but still flexible enough allowing the systems' future modifications (Olson 1998). This two-fold abstractness is one of the key characteristics of patterns in organizational context, for it enables the patterns to be used for different purposes by different roles.

Different pattern presentation formats are suggested in literature (Alexander et al. 1977; Buschmann et al. 1996; Coplien 2000; Gamma et al. 1995) indicating that different formats can be useful also in practical software development: more extensive descriptions for technical roles and particular for novices, and brief overview for non-technical roles and experts, who already know the pattern but need support for their memory. In the literature, however, patterns are

mainly discussed from the developer's point of view,thus forgetting the different backgrounds, skills and needs of other roles in the software development team.

Various skills are required for understanding the pattern descriptions. Traditionally the patterns are targeted at people skilled with programming and UML-notation (Buschmann et al. 1996; Gamma et al. 1995). A basic assumption is that the principles of object-oriented design are known, although even a brilliant OO programmer may not understand OO design (Holub 2004). This makes studying patterns burdensome to someone unskilled in programming. However, as the previously described case study and literature indicates (Cline 1996), also other roles can benefit for patterns, for example project managers.

Different Roles and Their Languages

Organisations are composed of actors performing their roles. In order to have efficient software development teams, the software development teams are composed of roles with different skills (Curtis et al. 1988). The development projects typically include tasks where actors with non-technical skills are needed. In this paper, we will use the term non-technical role to indicate an actor whose primary tasks are business-oriented, not code-centered.

In order to ensure flexible and efficient software development, the communication between the roles must be fluent (Walz et al. 1993). Problems arise when roles have different viewpoints and vocabularies about the software under development. Software patterns are proposed to facilitate the creation of common language (Buschmann et al. 1996; Gamma et al. 1995), and thus to facilitate the knowledge transfer between the roles. Considering design knowledge management, communication between technical and non-technical roles is particularly needed to assure that enough time is allocated for the pattern related tasks. As we present in this paper, the approach to patterns may vary from role to role, according to their needs, interests and skills.

Considering software projects in the case company, we identified five key roles, which typically formed a development team. The roles are architect, designer, programmer, project manager and product manager. The three first are seen as technical roles and two latter as non-technical ones. In the figure 1 the development team is illustrated in the context of their task technicality.

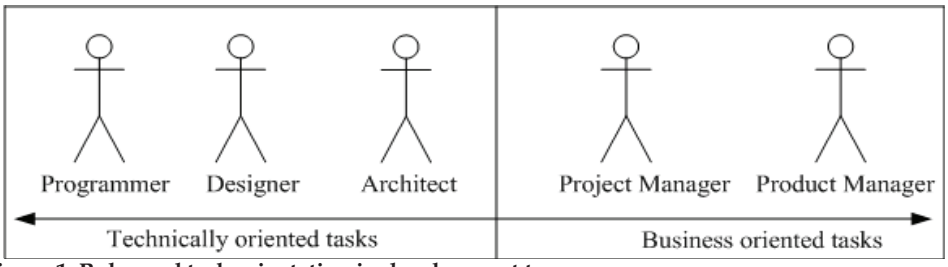

Figure 1. Roles and task orientation in development team.

The case company has several architects, who are responsible for the software architecture, including definitions of central design patterns. As a technical role, the architect often has the best knowledge of the possible system restrictions to the software. Architect is able to read and draw specifications using graphical notation.

Designers and programmers are often called as software engineers or developers. A designer has a responsibility of designing entire modules, and he specifies the detailed key technical issues of the source and test code. Programmer, in turn, creates the source code of the software. In the case company the practices set very few requirements on source code, hence an effective technical implementation is dependent on the developers' personal skills. Designer is able to read and draw graphical specifications and has programming skills. Programmer is able

to read the given specifications and his responsibility is to implement the product with given instructions. Although in practice the roles of designer and programmer are often performed by one actor, we prefer to handle them as separate roles, in order to clarify the different tasks and responsibilities of these roles and to clarify the different skills that are needed in their tasks.

Project managers' tasks in the company are to create and update the project plan. Skills in resource management and risk management are central. He is familiar with the business domain and can read graphical specifications, but is not interested in technical details or latest programming practices. In the case company, evert product has a product manager with a business approach. The tasks include executing and controlling the planned product activities. Technical issues can be rather uninteresting to this role and skills concerning for example reading UML notations can be limited.

The interests of the roles may vary greatly. Non-technical roles, meaning product and project managers, are mainly interested in if the product fills the requirements, how it is used in real life and when it can be delivered. The technical role, in turn, is interested for example in technologies that can be used and the trade-offs of each alternative design.

Differences in roles' skills and their varying interests have a significant effect on roles' abilities and willingness to understand and exploit patterns as design knowledge management means. Hence, pattern presentation formats can be tailored to be more suitable for a wider audience, thus making the patterns more understandable and more useful for the non-technical roles as well.

Formats of Pattern Presentation

There are three pattern presentation formats commonly used in literature: Alexandrian, GoF and Buschmann's formats (Alexander et al. 1977; Buschmann et al. 1996; Gamma et al. 1995), illustrated in figure 2. Most pattern presentation formats include three basic parts: the problem, its context and the proposed solution, although some other groupings exist as well (Cechich et al. 1999; Coplien 2000; Soundarajan et al. 2004).

Alexandrian format:	GoF format:	Buschmann format:
Title	Pattern name and	Name
Picture	classification	Also known as
Introduction	Intent	Example
Headline	Also known as	Context
Body	Motivation	Solution
Solution	Applicability	Structure
Diagram	Structure	Dynamics
Related patterns	Participants	Implementation
	Collaborations	Example resolved
	Consequences	Variants
	Implementation	Known uses
	Sample Code	Consequences
	Known Uses	See also
	Related Patterns	

Figure 2. Common pattern presentation formats.

Regardless of the presentation format, pattern Name is the most important section (Coplien 2000). Names distinct patterns from each other and enable pattern comparisons. By name the whole pattern content can be discussed without explaining the details, thus pattern names can be a part of design vocabulary. One pattern can have several names, presented in Also known as -section.

The Motivation provides a practical example of patterns context, problem and solution. The purpose is to give an overview of the area that pattern concerns. It usually is followed by an explanatory text. An example of descriptive picture is illustrated in figure 3, where the Observer –pattern is described as a single picture. The text then describes how the information in different objects can be updated by using an observer to mediate the changes (Gamma et al. 1995).

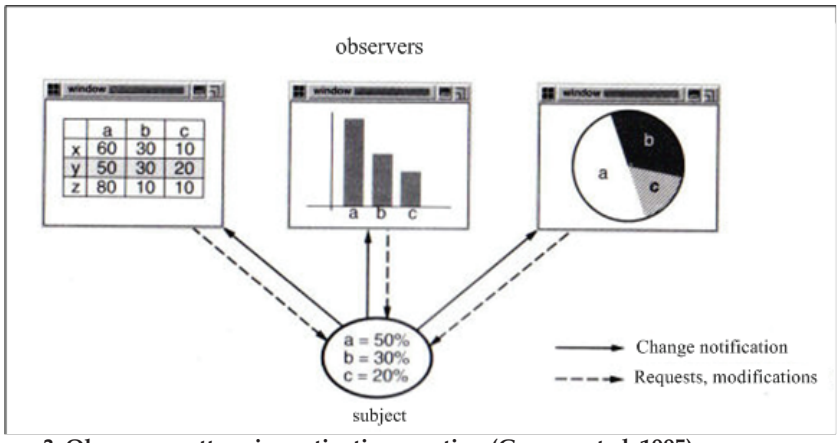

Figure 3. Observer -pattern in motivation –section (Gamma et al. 1995).

The Intent summarizes the general problem and introduces the solution in brief form, while the context of the problem addressed by the pattern is presented in the Context or Motivation sections. The context describes the factors that must be present for the pattern to work, for example programming language, size or scope. The Applicability describes situations where the pattern may be adopted; in a same manner the problem declares the details in the problem area. A concise problem statement helps the problem solver to decide whether to keep on reading this particular pattern description (Coplien 2000). Participants are the classes or objects that play key roles in implementing the function that the pattern describes. The Structure, in turn, includes the collaborations of the participants and presents them in graphical notation, as illustrated in the following figure 4.

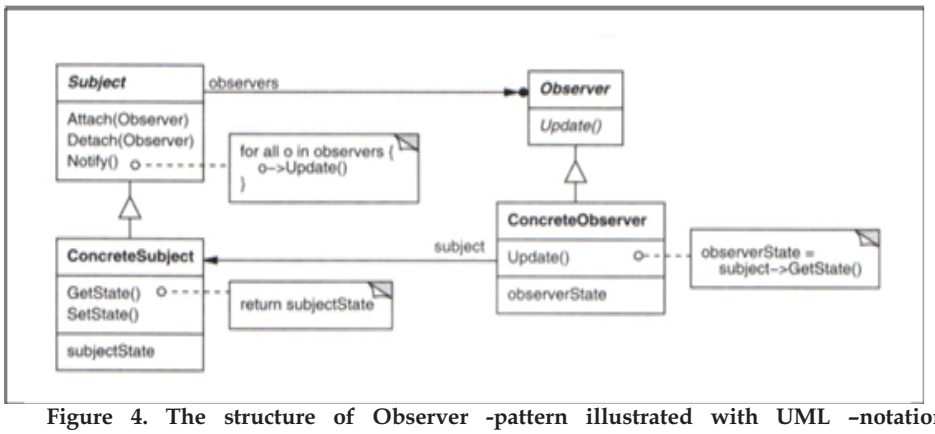

Figure 4. The structure of Observer -pattern illustrated with UML –notation (Gamma et al. 1995).

In the Dynamics or Collaborations sections, the participants' behavior is described in more detail, emphasizing the interactions of the pattern. Sequence charts are commonly used to

describe the behavior and timely interactions of each object, as illustrated in the following figure 5.

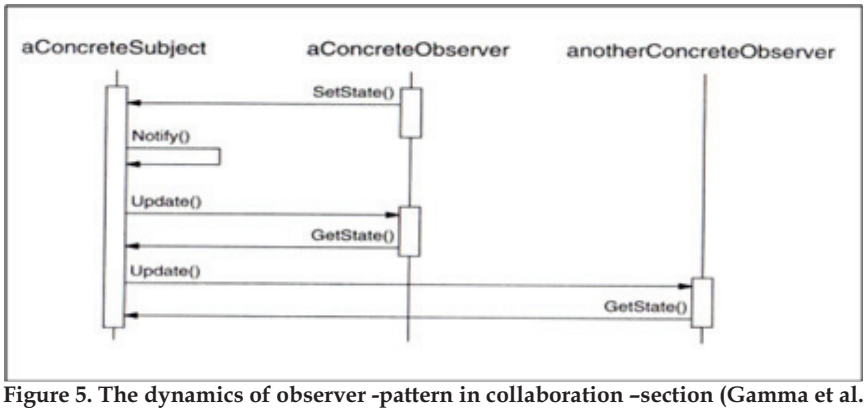

Figure 5. The dynamics of observer -pattern in collaboration –section (Gamma et al. 1995).

The Implementation section facilitates implementing the pattern in practice. Separate code examples are given in the Sample code or Example resolved sections. The Consequences section descibes the benefits and trade-offs which the pattern adoption brings along. The Known uses section, in turn, describes previous successful adoptions, thus facilitating the evaluation of the appropriateness of the pattern. The Related patterns and See also sections introduce other similar patterns, which can be used for the current problem. Browsing through other possible patterns may provide information about the pattern context and applicability, and suggest suitable pattern combinations.

The GoF pattern presentation format is particularly intended to help users in creating solutions to problems (Beck et al. 1994). The users focus less on when to apply the pattern and more on the actual structure and dynamics of the pattern itself. Thus, this pattern format is more descriptive rather than generative (Beck et al. 1994). Therefore, the GoF format is selected for analyzing the contents of the presentation format.

Analysis of Pattern Presentations

The purpose of each section in the pattern description is not always unambiguous. This is illustrated in figure 6 where the sections of GoF presentation format are depicted according to the Alexandrian view of pattern as a three part rule, including the context, the problem and the solution.

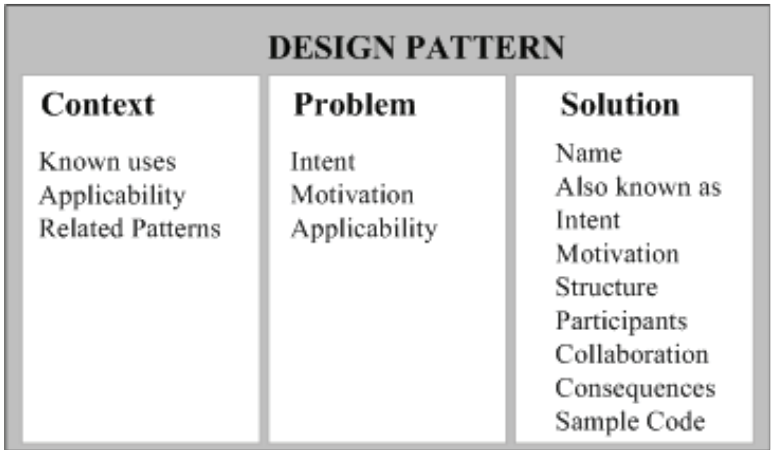

Figure 6. GoF presentation format as a three part rule.

The pattern sections Applicability, Intent and Motivation are present in more than one part, which enables logical connections between the parts. The applicability section is used both in finding the right context of the pattern, as well as in identifying the similarity between the problem areas of the current situation and the one described in the pattern description. Correspondingly, the intent section is used to identify the suitable problem area, but it simultaneously connects the problem area with possible solution by summarizing the purpose of the pattern. This connection is provided also by motivation section, but in a more structural way. Together the applicability, intent and motivation sections provide a consistent path from context to solution, which is essential particularly when learning the pattern for the first time.

The related patterns section is placed to the context part, for related patterns give the adopter hints about the environment where the pattern can be used. On the other hand, it could be placed to the solution part, for related patterns are links to the pattern system, thus facilitating the navigation in the pattern system and helping to understand the entity of patterns. These links between patterns describe specific pattern combinations which may serve as a solution. The pattern name is even more problematic, for there are no rules of how to give a name to a new (in-house) pattern, since the name's main purpose is to be informative and descriptive.

Pattern Parts in Design Knowledge Management

The results of the case study reflected that indeed reuse was a knowledge management problem, which could not be solved only by raising the abstraction level. This is indicated also by Highsmith (2003). Considering the role of project managers, the case findings indicated that design knowledge management was not supported by the processes, which is reflected also by the findings of Komi-Sirviö et. al (2002).

Considering pattern presentation from the design knowledge management perspective, the likely use of patterns depends on the roles in the development team. The relationships between the roles and the pattern use are illustrated in figure 8. The intensity of the arrow indicates the significance of different pattern parts to each role. The intention is to present the most essential patterns parts regarding to the tasks of the roles, without excluding others.

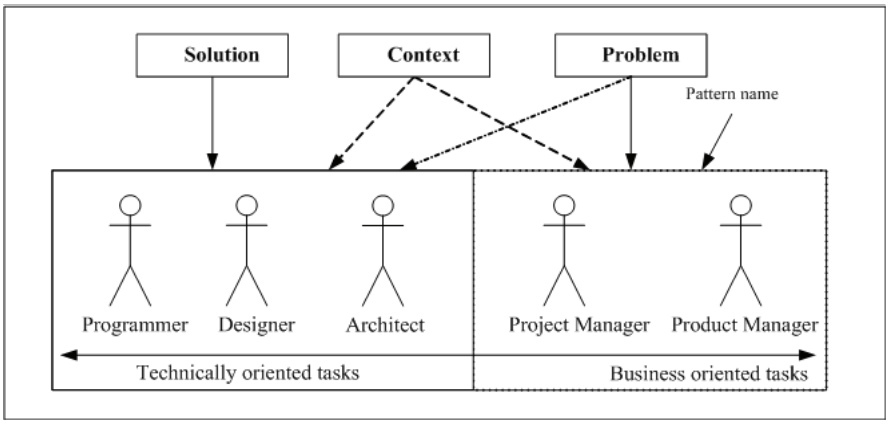

Figure 7. Essential pattern parts and development team roles

Some of the sections in pattern descriptions rely heavily on code, thus programming skills in appropriate language are needed for learning and adopting the pattern in a design. Particularly the sections, which are meant to facilitate the pattern adoption and making the right modifications, for example sections "implementation" and "sample code", often heavily employ source code in a specific programming language.

Many designers and programmers prefer software-centric visual aids such as class models and interaction graphs (Schmidt et al. 1996). Therefore, many pattern description formats use popular notations (such as Booch models and OMT) to express the structure and dynamic behavior concisely (Schmidt et al. 1996). This highlights the importance of graphical designing language knowledge.

The technically oriented roles need to understand the structure, participants and the collaborations most profoundly, and thus the solution part seems to be the most valuable in their work. The solution part is often described with programming language and graphical notation which are familiar and useful for these roles. For this particular reason, the business oriented roles may not be able to interpret the solution part. It should be mentioned that, while the context and the problem parts are of little (if any) interest to the programmer role, they are important for the designer and especially for the architect.

The non-technical roles need merely the problem part and the pattern name in their work. The motivation section gives managers a good overview of the proposed pattern, without going too much into details. Additionally, the motivation is often described with plain text and explaining pictures (Gamma et al. 1995), instead of graphical notations or programming languages, which facilitates the understanding.

It should be noted that, considering the roles in general level, the managers often are not interested on problems, but on solutions (Kolb 1996). However, in the case of patterns the aim is not to make the managers learn the pattern contents in detail, but to facilitate their understanding and communication about the design issues. The problem –part of pattern descriptions gives the best overview of the pattern, and therefore it seems to best suit for the non-technical roles. Furthermore, as indicated in (Curtis et al. 1988), gaining better understanding of design issues facilitates the project managers' actual task as well, namely estimating the needed time for development tasks.

As well technical as non-technical roles are able to use the context part, in order to support their thinking and decision making. Particularly the motivation section can be exploited by all roles, for it explains the pattern core in layman's terms, thus being both informative and abstract enough.

Thus, when creating an in-house pattern presentation format, the skills of the different roles should be acknowledged, and different parts of patterns should be described in an understandable way, not only for the technical people but also to the business oriented roles.

Concluding Remarks

In this paper the management of design knowledge is studied through software pattern presentation formats. The patterns are a tool to disseminate the design knowledge inside a company and a tool to facilitate the communication during software development. Furthermore, the pattern presentation formats can enable different readers to manage and exploit the knowledge captured in pattern format. The patterns described and analyzed in this paper were design patterns, but the principle of tailoring the pattern presentation formats can be adapted to the architectural patterns as well.

In software development, communication problems are likely to occur when managers have different viewpoint and vocabulary than the actual users of patterns, namely programmers, designers and architects. In order to facilitate the design knowledge management and to facilitate the communication of design issues, patterns should be understood by different roles, despite the differences of their skills. Meanwhile, pattern catalogues usually present patterns in a single format, not adjusted to different roles. Therefore, the proposed approach emphasizes dialogue between the individuals, as encouraged by DeSouza (2003).

A case study was made to motivate the research. The case revealed organizational factors related to patterns and their use, which should be acknowledged when patterns are introduced in an organization. Based on the case, the research question for this paper was stated as *"How to make the patterns more understandable for the business-oriented roles of software development?"* To address this question, pattern presentation was analyzed considering the possessed skills of different roles in software development. It was concluded that different roles can utilize different parts of pattern descriptions. Non-technical roles get best understanding of the design issues by using the problem part of the pattern description, while technical roles can better exploit the solution part.

However, the presentation format used in a company does not have to follow the formats that are present in the literature; rather, patterns can be formatted according to the organization's own needs and practices. It is up to the organization to decide what are seen important and which not, although some basic information, pattern name for example, should always be included, because of its functionality.

The role of a mentor can not be forgotten when promoting patterns in organizations (Beck et al. 1996). However, this role is hard to fulfill, unless the developers have no time to learn the patterns. We suggest that project managers can facilitate mentors' work by acknowledging the patterns and further by allocating time for the developers to study them.

Taken together, patterns' use needs organizational support as any other knowledge management activities. The support is hard to grant and implement, if the supported matter or its benefits are too burdensome to understand for the roles that make the actual decisions. Hence, to ensure patterns' organizational support, their use must be motivated for project managers and other managers as well.

Therefore, when introducing patterns as design knowledge management means for the managers, problem part can be emphasized to ensure the best understanding of the contents. Furthermore, a pattern presentation format employed in an organization can be adapted to the needs of a particular role to ensure a wider audience for the patterns. Technical tools can be utilized to automatically produce different views of a same pattern.

References

Alexander, C. The Timeless Way of Building, New York: Oxford University Press, 1979.

Alexander, C., Ishikawa, S., Silverstein, M., Jakobson, M., Fiksdahl-King, I., and Angel, S. A Pattern Language (Volume 2). New York: Oxford University press, 1977.

Beck, K., Crocker, R., Meszaros, G., Coplien, J.O., Dominick, L., Paulisch, F., and Vlissides, J. " Industrial experience with design patterns," 18th International Conference on Software Engineering, IEEE, Berlin, Germany, 1996, pp. 103-114.

Beck, K., and Johnson, R. "Patterns Generate Architectures," 8th European Conference of Object-Oriented Programming, Bologna, Italy, 1994.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. Pattern-oriented software architecture - A system of patterns, Chicester, West Sussex, England: John Wiley & Sons ltd., 1996.

Cechich, A., and Moore, R. "A Formal specification of GoF Design Patterns," The United Nations University, International Institute for Software Technology, 1999.

Cline, M.P. "The pros and cons of adopting and applying design patterns in the real world," Communications of the ACM (39:10), 1996, pp. 47-49.

Coplien, J. Software Patterns, New York: SIGS Books & Multimedia, 2000.

Curtis, B., Krasner, H., and Iscoe, N. "A field study of the software design process for large systems," Communications of the ACM (31:11), 1988, pp. 1268-1287.

DeSouza, K.C. "Barriers to Effective Use of Knowledge Management Systems in Software Engineering," Communications of the ACM (46:1), 2003, pp. 99-101.

Fayad, M., Schmidt, D.C., and Johnson, R. Building Application Frameworks, New York: John Wiley & Sons Inc., 1999.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. Design Patterns: Elements of reusable object oriented software, Boston, USA: Addison-Wesley, 1995.

Gomes, P., and Leitão, A. (eds.) A Tool for Management and Reuse of Software Design Knowledge, Berlin -Heidelberg: Springer Verlag, 2006.

Gross, D., and Yu, E. "From Non-Functional Requirements to Design through Patterns," Requirements Engineering (6), 2001.

Holub, A. Holub on Patterns: Learning design patterns by looking at code, USA: aPress, 2004.

Kolb, D.A. "Management and the learning process," in: How Organizations Learn, K. Starkey (ed.): International Thomson Publishing Company, London, 1996, pp. 270-287.

Komi-Sirviö, S., Mäntyniemi, A., and Seppänen, V. "Toward A Practical Solution for Capturing knowledge for Software Projects," IEEE Software (May/June), 2002, pp. 60-62.

Kucza, T., Nättinen, M. and Parviainen, P., "Improving knowledge management in software reuse process," PROFES, Springer-Verlag, Kaiserslautern, Germany, 2001, pp. 141-152.

Nonaka, I. "A Dynamic Theory of Organizational Knowledge Creation," Organization science (5:1), 1994.

Olson, D. "FAQ from Bruce Anderson's group at ChiliPLoP'98," Wickenburg AZ, 17-20 March 1998, 1998.

Rus, I., and Lindvall, M. "Knowledge Management in Software Engineering," IEEE Software (19:3), 2002, pp. 26-38.

Schmidt, D.C., Fayad, M., Johnson, R.E., and Guest Editors "Software Patterns," Communications of the ACM (39:10), 1996.

Selfridge, P.G., Terveen, L.G., and Long, M.D. " Managing design knowledge to provide assistance to large-scalesoftware development," Knowledge-Based Software Engineering Conference, IEEE, McLean, VA, USA, 1992, pp. 163-170.

Soundarajan, N., and Hallstrom, J.O. "Responsibilities and Rewards: Specifying Design Patterns," 26th International Conference on Software Engineering, IEEE Computer Society, Edinburgh, Scotland, 2004.

Stein, E.W. "Organizational Memory: Review of Concepts and Recommendations for Management," International Journal of Information Management (15:1), 1995, pp. 17-32.

Walz, D.B., Elam, J.J., and Curtis, B. "Inside Software Design Team: Knowledge Acquisition, Sharing and Integration," Communications of the ACM (36:10), 1993, pp. 63-77.

IV

DESIGN PATTERNS AND ORGANIZATIONAL MEMORY IN MOBILE APPLICATION DEVELOPMENT

by

Riikka Ahlgren & Jouni Markkula, 13.6.2005

Product Focused Software Engineering. Lecture Notes in Computer Science, Vol. 3547/2005, 143-156

V

APPLYING PATTERNS FOR IMPROVING SUBCONTRACTING MANAGEMENT

by

Riikka Ahlgren, Jari Penttilä & Jouni Markkula, 31.10.2005

On the Move to Meaningful Internet Systems 2005: OTM Workshops
Lecture Notes in Computer Science, 2005, Volume 3762/2005, 572-581

JYVÄSKYLÄ STUDIES IN COMPUTING

1 Ropponen, Janne, Software risk management - foundations, principles and empirical findings. 273 p. Yhteenveto 1 p. 1999.
2 Kuzmin, Dmitri, Numerical simulation of reactive bubbly flows. 110 p. Yhteenveto 1 p. 1999.
3 Karsten, Helena, Weaving tapestry: collaborative information technology and organisational change. 266 p. Yhteenveto 3 p. 2000.
4 Koskinen, Jussi, Automated transient hypertext support for software maintenance. 98 p. (250 p.) Yhteenveto 1 p. 2000.
5 Ristaniemi, Tapani, Synchronization and blind signal processing in CDMA systems. - Synkronointi ja sokea signaalinkäsittely CDMA järjestelmässä. 112 p. Yhteenveto 1 p. 2000.
6 Laitinen, Mika, Mathematical modelling of conductive-radiative heat transfer. 20 p. (108 p.) Yhteenveto 1 p. 2000.
7 Koskinen, Minna, Process metamodelling. Conceptual foundations and application. 213 p. Yhteenveto 1 p. 2000.
8 Smolianski, Anton, Numerical modeling of two-fluid interfacial flows. 109 p. Yhteenveto 1 p. 2001.
9 Nahar, Nazmun, Information technology supported technology transfer process. A multi-site case study of high-tech enterprises. 377 p. Yhteenveto 3 p. 2001.
10 Fomin, Vladislav V., The process of standard making. The case of cellular mobile telephony. - Standardin kehittämisen prosessi. Tapaustutkimus solukkoverkkoon perustuvasta matkapuhelintekniikasta. 107 p. (208 p.) Yhteenveto 1 p. 2001.
11 Päivärinta, Tero, A genre-based approach to developing electronic document management in the organization. 190 p. Yhteenveto 1 p. 2001.
12 Häkkinen, Erkki, Design, implementation and evaluation of neural data analysis environment. 229 p. Yhteenveto 1 p. 2001.
13 Hirvonen, Kullervo, Towards better employment using adaptive control of labour costs of an enterprise. 118 p. Yhteenveto 4 p. 2001.
14 Majava, Kirsi, Optimization-based techniques for image restoration. 27 p. (142 p.) Yhteenveto 1 p. 2001.
15 Saarinen, Kari, Near infra-red measurement based control system for thermo-mechanical refiners. 84 p. (186 p.) Yhteenveto 1 p. 2001.
16 Forsell, Marko, Improving component reuse in software development. 169 p. Yhteenveto 1 p. 2002.
17 Virtanen, Pauli, Neuro-fuzzy expert systems in financial and control engineering. 245 p. Yhteenveto 1 p. 2002.
18 Kovalainen, Mikko, Computer mediated organizational memory for process control. Moving CSCW research from an idea to a product. 57 p. (146 p.) Yhteenveto 4 p. 2002.
19 Hämäläinen, Timo, Broadband network quality of service and pricing. 140 p. Yhteenveto 1 p. 2002.
20 Martikainen, Janne, Efficient solvers for discretized elliptic vector-valued problems. 25 p. (109 p.) Yhteenveto 1 p. 2002.
21 Mursu, Anja, Information systems development in developing countries. Risk management and sustainability analysis in Nigerian software companies. 296 p. Yhteenveto 3 p. 2002.
22 Seleznyov, Alexandr, An anomaly intrusion detection system based on intelligent user recognition. 186 p. Yhteenveto 3 p. 2002.
23 Lensu, Anssi, Computationally intelligent methods for qualitative data analysis. 57 p. (180 p.) Yhteenveto 1 p. 2002.
24 Ryabov, Vladimir, Handling imperfect temporal relations. 75 p. (145 p.) Yhteenveto 2 p. 2002.
25 Tsymbal, Alexey, Dynamic integration of data mining methods in knowledge discovery systems. 69 p. (170 p.) Yhteenveto 2 p. 2002.
26 Akimov, Vladimir, Domain decomposition methods for the problems with boundary layers. 30 p. (84 p.). Yhteenveto 1 p. 2002.
27 Seyukova-Rivkind, Ludmila, Mathematical and numerical analysis of boundary value problems for fluid flow. 30 p. (126 p.) Yhteenveto 1 p. 2002.
28 Hämäläinen, Seppo, WCDMA Radio network performance. 235 p. Yhteenveto 2 p. 2003.
29 Pekkola, Samuli, Multiple media in group work. Emphasising individual users in distributed and real-time CSCW systems. 210 p. Yhteenveto 2 p. 2003.
30 Markkula, Jouni, Geographic personal data, its privacy protection and prospects in a location-based service environment. 109 p. Yhteenveto 2 p. 2003.
31 Honkaranta, Anne, From genres to content analysis. Experiences from four case organizations. 90 p. (154 p.) Yhteenveto 1 p. 2003.
32 Raitamäki, Jouni, An approach to linguistic pattern recognition using fuzzy systems. 169 p. Yhteenveto 1 p. 2003.
33 Saalasti, Sami, Neural networks for heart rate time series analysis. 192 p. Yhteenveto 5 p. 2003.
34 Niemelä, Marketta, Visual search in graphical interfaces: a user psychological approach. 61 p. (148 p.) Yhteenveto 1 p. 2003.
35 You, Yu, Situation Awareness on the world wide web. 171 p. Yhteenveto 2 p. 2004.
36 Taatila, Vesa, The concept of organizational competence – A foundational analysis. - Perusteanalyysi organisaation kompetenssin käsitteestä. 111 p. Yhteenveto 2 p. 2004.

37 LYYTIKÄINEN, VIRPI, Contextual and structural metadata in enterprise document management. - Konteksti- ja rakennemetatieto organisaation dokumenttien hallinnassa. 73 p. (143 p.) Yhteenveto 1 p. 2004.
38 KAARIO, KIMMO, Resource allocation and load balancing mechanisms for providing quality of service in the Internet. 171 p. Yhteenveto 1 p. 2004.
39 ZHANG, ZHEYING, Model component reuse. Conceptual foundations and application in the metamodeling-based systems analysis and design environment. 76 p. (214 p.) Yhteenveto 1 p. 2004.
40 HAARALA, MARJO, Large-scale nonsmooth optimization variable metric bundle method with limited memory. 107 p. Yhteenveto 1 p. 2004.
41 KALVINE, VIKTOR, Scattering and point spectra for elliptic systems in domains with cylindrical ends. 82 p. 2004.
42 DEMENTIEVA, MARIA, Regularization in multistage cooperative games. 78 p. 2004.
43 MAARANEN, HEIKKI, On heuristic hybrid methods and structured point sets in global continuous optimization. 42 p. (168 p.) Yhteenveto 1 p. 2004.
44 FROLOV, MAXIM, Reliable control over approximation errors by functional type a posteriori estimates. 39 p. (112 p.) 2004.
45 ZHANG, JIAN, Qos- and revenue-aware resource allocation mechanisms in multiclass IP networks. 85 p. (224 p.) 2004.
46 KUJALA, JANNE, On computation in statistical models with a psychophysical application. 40 p. (104 p.) 2004.,
47 SOLBAKOV, VIATCHESLAV, Application of mathematical modeling for water environment problems. 66 p. (118 p.) 2004.
48 HIRVONEN, ARI P., Enterprise architecture planning in practice. The Perspectives of information and communication technology service provider and end-user. 44 p. (135 p.) Yhteenveto 2 p. 2005.
49 VARTIAINEN, TERO, Moral conflicts in a project course in information systems education. 320 p. Yhteenveto 1p. 2005.
50 HUOTARI, JOUNI, Integrating graphical information system models with visualization techniques. - Graafisten tietojärjestelmäkuvausten integrointi visualisointitekniikoilla. 56 p. (157 p.) Yhteenveto 1p. 2005.
51 WALLENIUS, EERO R., Control and management of multi-access wireless networks. 91 p. (192 p.) Yhteenveto 3 p. 2005.
52 LEPPÄNEN, MAURI, An ontological framework and a methodical skeleton for method engineering – A contextual approach. 702 p. Yhteenveto 2 p. 2005.
53 MATYUKEVICH, SERGEY, The nonstationary Maxwell system in domains with edges and conical points. 131 p. Yhteenveto 1 p. 2005.
54 SAYENKO, ALEXANDER, Adaptive scheduling for the QoS supported networks. 120 p. (217 p.) 2005.
55 KURJENNIEMI, JANNE, A study of TD-CDMA and WCDMA radio network enhancements. 144 p. (230 p.) Yhteenveto 1 p. 2005.
56 PECHENIZKIY, MYKOLA, Feature extraction for supervised learning in knowledge discovery systems. 86 p. (174 p.) Yhteenveto 2 p. 2005.
57 IKONEN, SAMULI, Efficient numerical methods for pricing American options. 43 p. (155 p.) Yhteenveto 1 p. 2005.
58 KÄRKKÄINEN, KARI, Shape sensitivity analysis for numerical solution of free boundary problems. 83 p. (119 p.) Yhteenveto 1 p. 2005.
59 HELFENSTEIN, SACHA, Transfer. Review, reconstruction, and resolution. 114 p. (206 p.) Yhteenveto 2 p. 2005.
60 NEVALA, KALEVI, Content-based design engineering thinking. In the search for approach. 64 p. (126 p.) Yhteenveto 1 p. 2005.
61 KATASONOV, ARTEM, Dependability aspects in the development and provision of location-based services. 157 p. Yhteenveto 1 p. 2006.
62 SARKKINEN, JARMO, Design as discourse: Representation, representational practice, and social practice. 86 p. (189 p.) Yhteenveto 1 p. 2006.
63 ÄYRÄMÖ, SAMI, Knowledge mining using robust clustering. 296 p. Yhteenveto 1 p. 2006.
64 IFINEDO, PRINCELY EMILI, Enterprise resource planning systems success assessment: An integrative framework. 133 p. (366 p.) Yhteenveto 3 p. 2006.
65 VIINIKAINEN, ARI, Quality of service and pricingin future multiple service class networks. 61 p. (196 p.) Yhteenveto 1 p. 2006.
66 WU, RUI, Methods for space-time parameter estimation in DS-CDMA arrays. 73 p. (121 p.) 2006.
67 PARKKOLA, HANNA, Designing ICT for mothers. User psychological approach. – Tieto- ja viestintätekniikoiden suunnittelu äideille. Käyttäjäpsykologinen näkökulma. 77 p. (173 p.) Yhteenveto 3 p. 2006.
68 HAKANEN, JUSSI, On potential of interactive multiobjective optimization in chemical process design. 75 p. (160 p.) Yhteenveto 2 p. 2006.
69 PUTTONEN, JANI, Mobility management in wireless networks. 112 p. (215 p.) Yhteenveto 1 p. 2006.
70 LUOSTARINEN, KARI, Resource , management methods for QoS supported networks. 60 p. (131 p.) 2006.
71 TURCHYN, PAVLO, Adaptive meshes in computer graphics and model-based simulation. 27 p. (79 p.) Yhteenveto 1 p.
72 ZHOVTOBRYUKH, DMYTRO, Context-aware web service composition. 290 p. Yhteenveto 2 p. 2006.

73 KOHVAKKO, NATALIYA, Context modeling and utilization in heterogeneous networks. 154 p. Yhteenveto 1 p. 2006.
74 MAZHELIS, OLEKSIY, Masquerader detection in mobile context based on behaviour and environment monitoring. 74 p. (179 p). Yhteenveto 1 p. 2007.
75 SILTANEN, JARMO, Quality of service and dynamic scheduling for traffic engineering in next generation networks. 88 p. (155 p.) 2007.
76 KUUVA, SARI, Content-based approach to experiencing visual art. - Sisältöperustainen lähestymistapa visuaalisen taiteen kokemiseen. 203 p. Yhteenveto 3 p. 2007.
77 RUOHONEN, TONI, Improving the operation of an emergency department by using a simulation model. 164 p. 2007.
78 NAUMENKO, ANTON, Semantics-based access control in business networks. 72 p. (215 p.) Yhteenveto 1 p. 2007.
79 WAHLSTEDT, ARI, Stakeholders' conceptions of learning in learning management systems development. - Osallistujien käsitykset oppimisesta oppimisympäristöjen kehittämisessä. 83 p. (130 p.) Yhteenveto 1 p. 2007.
80 ALANEN, OLLI, Quality of service for triple play services in heterogeneous networks. 88 p. (180 p.) Yhteenveto 1 p. 2007.
81 NERI, FERRANTE, Fitness diversity adaptation in memetic algorithms. 80 p. (185 p.) Yhteenveto 1 p. 2007.
82 KURHINEN, JANI, Information delivery in mobile peer-to-peer networks. 46 p. (106 p.) Yhteenveto 1 p. 2007.
83 KILPELÄINEN, TURO, Genre and ontology based business information architecture framework (GOBIAF). 74 p. (153 p.) Yhteenveto 1 p. 2007.
84 YEVSEYEVA, IRYNA, Solving classification problems with multicriteria decision aiding approaches. 182 p. Yhteenveto 1 p. 2007.
85 KANNISTO, ISTO, Optimized pricing, QoS and segmentation of managed ICT services. 45 p. (111 p.) Yhteenveto 1 p. 2007.
86 GORSHKOVA, ELENA, A posteriori error estimates and adaptive methods for incompressible viscous flow problems. 72 p. (129 p.) Yhteenveto 1 p. 2007.
87 LEGRAND, STEVE, Use of background real-world knowledge in ontologies for word sense disambiguation in the semantic web. 73 p. (144 p.) Yhteenveto 1 p. 2008.
88 HÄMÄLÄINEN, NIINA, Evaluation and measurement in enterprise and software architecture management. - Arviointi ja mittaaminen kokonais- ja ohjelmistoarkkitehtuurien hallinnassa. 91 p. (175 p.) Yhteenveto 1 p. 2008.
89 OJALA, ARTO, Internationalization of software firms: Finnish small and medium-sized software firms in Japan. 57 p. (180 p.) Yhteenveto 2 p. 2008.
90 LAITILA, ERKKI, Symbolic Analysis and Atomistic Model as a Basis for a Program Comprehension Methodology. 321 p. Yhteenveto 3 p. 2008.
91 NIHTILÄ, TIMO, Performance of Advanced Transmission and Reception Algorithms for High Speed Downlink Packet Access. 93 p. (186 p.) Yhteenveto 1 p. 2008.
92 SETÄMAA-KÄRKKÄINEN, ANNE, Network connection selection-solving a new multiobjective optimization problem. 52 p. (111p.) Yhteenveto 1 p. 2008.
93 PULKKINEN, MIRJA, Enterprise architecture as a collaboration tool. Discursive process for enterprise architecture management, planning and development. 130 p. (215 p.) Yhteenveto 2 p. 2008.
94 PAVLOVA, YULIA, Multistage coalition formation game of a self-enforcing international environmental agreement. 127 p. Yhteenveto 1 p. 2008.
95 NOUSIAINEN, TUULA, Children's involvement in the design of game-based learning environments. 297 p. Yhteenveto 2 p. 2008.
96 KUZNETSOV, NIKOLAY V., Stability and oscillations of dynamical systems. Theory and applications. 116 p. Yhteenveto 1 p. 2008.
97 KHRIYENKO, OLEKSIY, Adaptive semantic Web based environment for web resources. 193 p. Yhteenveto 1 p. 2008.
98 TIRRONEN, VILLE, Global optimization using memetic differential evolution with applications to low level machine vision. 98 p. (248 p.) Yhteenveto 1 p. 2008.
99 VALKONEN, TUOMO, Diff-convex combinations of Euclidean distances: A search for optima. 148 p. Yhteenveto 1 p. 2008.
100 SARAFANOV, OLEG, Asymptotic theory of resonant tunneling in quantum waveguides of variable cross-section. 69 p. Yhteenveto 1 p. 2008.
101 POZHARSKIY, ALEXEY, On the electron and phonon transport in locally periodical waveguides. 81 p. Yhteenveto 1 p. 2008.
102 AITTOKOSKI, TIMO, On challenges of simulation-based globaland multiobjective optimization. 80 p. (204 p.) Yhteenveto 1 p. 2009.
103 YALAHO, ANICET, Managing offshore outsourcing of software development using the ICT-supported unified process model: A cross-case analysis. 91 p. (307 p.) Yhteenveto 4 p. 2009.
104 KOLLANUS, SAMI, Tarkastuskäytänteiden kehittäminen ohjelmistoja tuottavissa organisaatioissa. - Improvement of inspection practices in software organizations. 179 p. Summary 4 p. 2009.
105 LEIKAS, JAANA, Life-Based Design. 'Form of life' as a foundation for ICT design for older adults. - Elämälähtöinen suunnittelu. Elämänmuoto ikääntyville tarkoitettujen ICT tuotteiden ja palvelujen suunnittelun lähtökohtana. 218 p. (318 p.) Yhteenveto 4 p. 2009.

106 Vasilyeva, Ekaterina, Tailoring of feedback in web-based learning systems: Certitude-based assessment with online multiple choice questions. 124 p. (184 p.) Yhteenveto 2 p. 2009.

107 Kudryashova, ElenaV., Cycles in continuous and discrete dynamical systems. Computations, computer assisted proofs, and computer experiments. 79 p. (152 p.) Yhteenveto 1 p. 2009.

108 Blackledge, Jonathan, Electromagnetic scattering and inverse scattering solutions for the analysis and processing of digital signals and images. 297 p. Yhteenveto 1 p. 2009.

109 Ivannikov, Andriy, Extraction of event-related potentials from electroencephalography data. - Herätepotentiaalien laskennallinen eristäminen EEG-havaintoaineistosta. 108 p. (150 p.) Yhteenveto 1 p. 2009.

110 kalyakin, Igor, Extraction of mismatch negativity from electroencephalography data. - Poikkeavuusnegatiivisuuden erottaminen EEG-signaalista. 47 p. (156 p.) Yhteenveto 1 p. 2010.

111 Heikkilä, Marikka, Coordination of complex operations over organisational boundaries. 265 p. Yhteenveto 3 p. 2010.

112 Fekete, Gábor, Network interface management in mobile and multihomed nodes. 94 p. (175 p.) Yhteenveto 1 p. 2010.

113 Kujala, Tuomo, Capacity, workload and mental contents - Exploring the foundations of driver distraction. 146 p. (253 p.) Yhteenveto 2 p. 2010.

114 Lugano, Giuseppe, Digital community design - Exploring the role of mobile social software in the process of digital convergence. 253 p. (316 p.) Yhteenveto 4 p. 2010.

115 Kampylis, Panagiotis, Fostering creative thinking. The role of primary teachers. - Luovaa ajattelua kehittämässä. Alakoulun opettajien rooli. 136 p. (268 p.) Yhteenveto 2 p. 2010.

116 Toivanen, Jukka, Shape optimization utilizing consistent sensitivities. - Muodon optimointi käyttäen konsistentteja herkkyyksiä. 55 p. (130p.) Yhteenveto 1 p. 2010.

117 Mattila, Keijo, Implementation techniques for the lattice Boltzmann method. - Virtausdynamiikan tietokonesimulaatioita Hila-Boltzmann -menetelmällä: implementointi ja reunaehdot. 177 p. (233 p.) Yhteenveto 1 p. 2010.

118 Cong, Fengyu, Evaluation and extraction of mismatch negativity through exploiting temporal, spectral, time-frequency, and spatial features. - Poikkeavuusnegatiivisuuden (MMN) erottaminen aivosähkönauhoituksista käyttäen ajallisia, spektraalisia, aikataajuus - ja tilapiirteitä. 57 p. (173 p.) Yhteenveto 1 p. 2010.

119 Liu, Shenghua, Interacting with intelligent agents. Key issues in agent-based decision support system design. 90 p. (143 p.) Yhteenveto 2 p. 2010.

120 Airaksinen, Tuomas, Numerical methods for acoustics and noise control. - Laskennallisia menetelmiä akustisiin ongelmiin ja melunvaimennukseen. 58 p. (133 p.) Yhteenveto 2 p. 2010.

121 Weber, Matthieu, Parallel global optimization Structuring populations in differential evolution. - Rinnakkainen globaalioptimointi. Populaation rakenteen määrittäminen differentiaalievoluutiossa. 70 p. (185 p.) Yhteenveto 2 p. 2010.

122 Väärämäki, Tapio, Next generation networks, mobility management and appliances in intelligent transport systems. - Seuraavan sukupolven tietoverkot, liikkuvuuden hallinta ja sovellutukset älykkäässä liikenteessä. 50 p. (111 p.) Yhteenveto 1 p. 2010.

123 Viukari, Leena, Tieto- ja viestintätekniikka-välitteisen palvelun kehittämisen kolme diskurssia. - Three discourses for an ICT-service development . 304 p. Summary 5 p. 2010.

124 Puurtinen, Tuomas, Numerical simulation of low temperature thermal conductance of corrugated nanofibers. - Poimutettujen nanokuitujen lämmönjohtavuuden numeerinen simulointi matalissa lämpötiloissa . 114 p. Yhteenveto 1 p. 2010.

125 Hiltunen, Leena, Enhancing web course design using action research . - Verkko-opetuksen suunnittelun kehittäminen toimintatutkimuksen keinoin . 192 p. Yhteenveto 2 p. 2010.

126 Aho, Kari, Enhancing system level performance of third generation cellular networks through VoIP and MBMS services. 121 p. (221 p.). Yhteenveto 2 p. 2010.

127 Häkkinen, Markku, Why alarms fail. A cognitive explanatory model. 102 p. (210 p.). Yhteenveto 1 p. 2010.

128 Pennanen, Anssi, A graph-based multigrid with applications. - Graafipohjainen monihilamenetelmä sovelluksineen. 52 p. (128 p.). Yhteenveto 2 p. 2010.

129 Ahlgren, Riikka, Software patterns, organizational learning and software process improvement. 70 p. (137 p.). Yhteenveto 1 p. 2011.